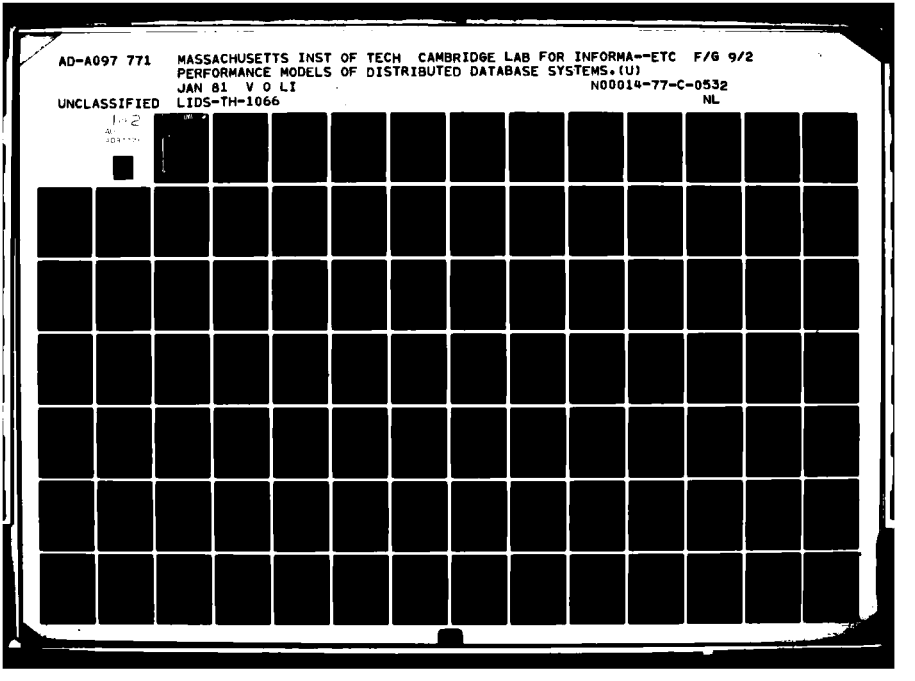
AD-A097 771 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR INFORMA--ETC F/G 9/2
PERFORMANCE MODELS OF DISTRIBUTED DATABASE SYSTEMS.(U)
JAN 81 V O LI N00014-77-C-0532
UNCLASSIFIED LIDS-TH-1066 NL

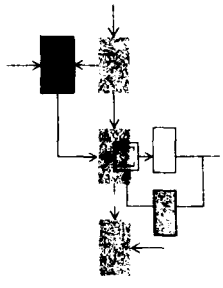


LEVEL II

January 1981

LIDS-TH-1066

Research Supported By:
ONR Contract N00014-77-C-0532



PERFORMANCE MODELS OF DISTRIBUTED DATABASE SYSTEMS

Victor On-Kwok Li

Laboratory for Information and Decision Systems
MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MASSACHUSETTS 02139

(11) Jan[redacted]981

(14) LIDS-TH-1066

(12) 1981

(6) PERFORMANCE MODELS OF DISTRIBUTED DATABASE SYSTEMS

by

(10) Victor On-Kwok/Li

(9) Doctor. thesis

This research is based on the unaltered thesis of Victor On-Kwok Li, submitted in partial fulfillment of the requirements for the degree of Doctor of Science at the Massachusetts Institute of Technology in January, 1981. The research was conducted at the M.I.T. Laboratory for Information and Decision Systems with support provided by the Office of Naval Research under contract ONR/N00014-77-C-0532.

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139

410950

PERFORMANCE MODELS OF DISTRIBUTED DATABASE SYSTEMS

by

VICTOR ON-KWOK LI

S.B., Massachusetts Institute of Technology
(1977)

S.M., Massachusetts Institute of Technology
(1979)

E.E., Massachusetts Institute of Technology
(1980)

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1981

Signature of Author .....Victor Li.....
Department of Electrical Engineering and Computer Science

Certified by ...William B. Davenport, Jr.....
Wilbur B. Davenport, Jr.
Thesis supervisor

Accepted by .....
Chairman, Departmental Committee on Graduate Students

PERFORMANCE MODELS OF DISTRIBUTED DATABASE SYSTEMS

by

VICTOR ON-KWOK LI

Submitted to the Department of Electrical Engineering and Computer Science
on January 26, 1981 in partial fulfillment of the
requirements for the Degree of Doctor of Science

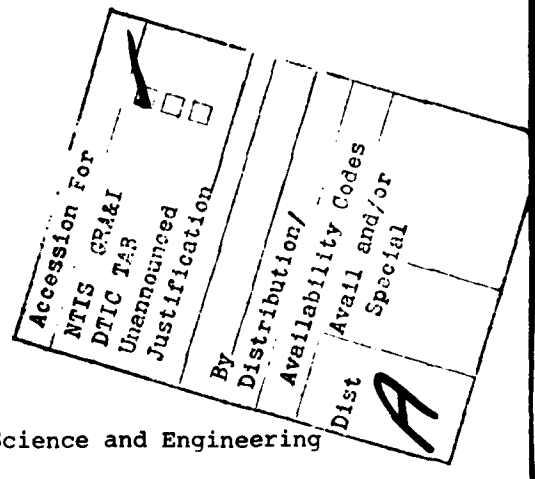
ABSTRACT

A distributed database (DDB) consists of redundant copies of data files geographically distributed on a computer network. This thesis develops a performance model of a DDB. This model can be used to compare the performance (i.e. response time, utilization, etc.) of different concurrency control algorithms.

We started by developing a network of queues model of a communication subnetwork. We have originally attempted to employ Jackson's Model but have concluded that Jackson's Model is inadequate for our purposes. The Independent Queues Model that we developed in this thesis makes somewhat stronger assumptions than Jackson's Model, but has more flexibility and approximates better a real communication subnetwork.

We found that in a general DDB, concurrency control algorithms could not be modelled accurately without taking into consideration the particular query processing strategy used. We have therefore developed two new query processing strategies: the MST and the MDT Algorithms. These two algorithms are easy to analyze and to implement.

We next modelled the competition among different transactions in the DDB for the services of the database management system. Probabilitic arguments were used to determine the probability of conflicts between different database transactions and the delay due to conflicts.



Thesis Supervisor: Dr. Wilbur B. Davenport, Jr.

Title: Professor of Communications Science and Engineering

ACKNOWLEDGEMENT

I wish to thank my thesis advisor, Prof. Wilbur B. Devenport, Jr. His guidance and advice have been most helpful during the course of my thesis research and throughout my graduate studies at M.I.T.

I am grateful to my thesis readers, Prof. Robert G. Gallager, Prof. Christos H. Papadimitriou and Prof. Frederick C. Hennie for their comments. Prof. Gallager has been especially helpful. Several of his suggestions led to results contained in this thesis.

I would also like to acknowledge assistance from Prof. Alvin Drake and Prof. Pierre Humblet.

Finally, I would like to thank my wife, Regina, for her understanding and encouragement during my graduate studies and for her typing of the thesis.

This research was carried out at the M.I.T. Laboratory for Information and Decision Systems with support provided by the Office of Naval Research under contract ONR/N00014-77-C-0532. (M.I.T. OSP No. 85552).

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This report describes a program of research to develop a performance model of a distributed database (DDB). The field of DDB is relatively new and growing rapidly. Numerous algorithms have been proposed for data retrieval, update synchronization and file allocation. This thesis develops a tool that will enable one to compare the performance (i.e., response time, throughput, utilization, etc.) of the different algorithms, and to propose better, more efficient solutions.

The approach will be to approximate the communication subnetwork by a Network of Queues Model. Probabilistic arguments will be used to specialize the model to accomodate the characteristics of a DDB.

1.1. Definitions

A database is a collection of operational data used by the application systems of some particular enterprise. The Database Management System (DMS) is the special software designed to provide each application with its own view of the common data, to implement operations for retrieval and update, and to resolve conflicts between concurrent users. The development of computer communication networks introduced the concept of the distributed database which is a database whose physical copies of data (often redundant) are distributed on a computer network. The Distributed Database Management System (DDMS) permits a collection of data relevant to a particular enterprise to be managed on a network of geographically dispersed computers (computer sites). Fig. 1.1 shows the basic architecture of a DDB. An arbitrary network of computer sites is connected by communication links. Attached to each computer site are the data files, sensors and terminals

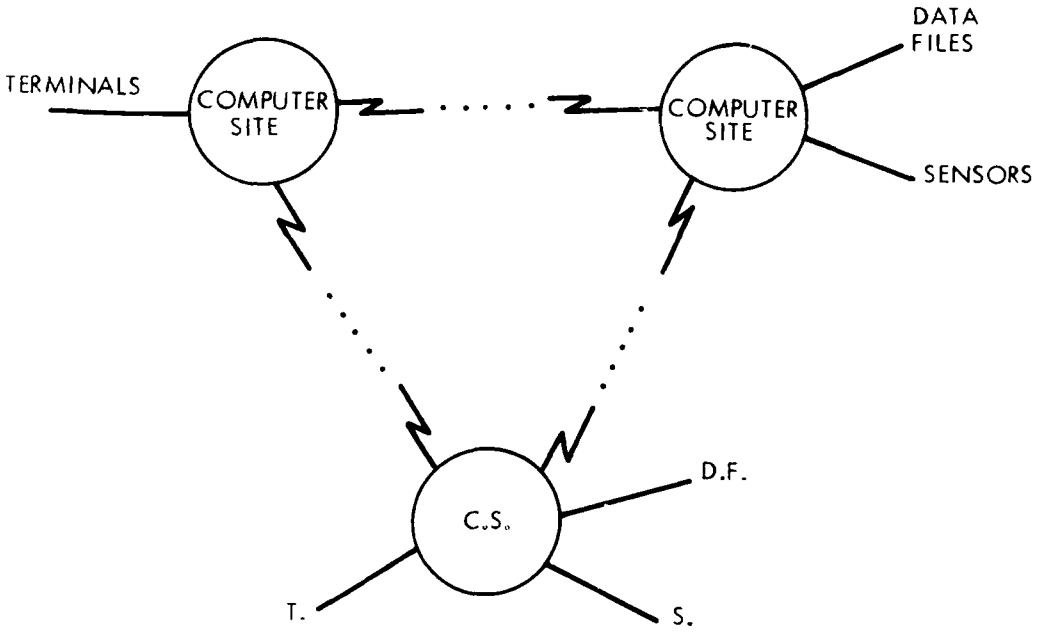


Figure 1.1 Basic Architecture of a Distributed Database

through which users can access the data.

1.2 Advantages and Disadvantages of DDB

Some enterprises, such as military Command, Control and Communications systems are distributed in nature; since command posts and sensory gathering points are geographically dispersed, users are necessarily dispersed. For example, the Navy has remote sensors and databases distributed all over the world. Other potential users are airline reservation systems, and electronic fund transfer systems. A typical user will be an enterprise which maintains operations at several geographically dispersed sites, and whose activities necessitate inter-site communication of data.

Distributed databases also offer the following advantages when compared to centralized databases:

(1) Improved throughput - the availability of multiple computers means that throughput can be increased via parallel processing.

(2) Sharing - geographically dispersed data and equipment can be shared.

(3) Modular expansion - distributed database systems can be expanded by the addition of new nodes (database sites) to the network.

For distributed databases with redundant copies of data, there are additional advantages:

(1) Improved reliability/survivability - through redundancy of data.

(2) Improved response time - by storing data in locations where it is frequently read. Communication delay is reduced since files which are under heavy demand in several geographically dispersed locations can be stored redundantly. However, more redundant copies also means increased delay during writes.

There are two major implementation problems associated with distributed

databases. The first problem is that communication channels between sites are often very slow compared to the storage devices at the local computer sites. For example, the ARPANET can move data at about 25 Kbps (kilobits/sec) while standard disks can move data at about 1 Mbps (megabits/sec), a 40-fold increase in rate. Besides, networks have relatively long access times, corresponding to the propagation delay for one message to go from one computer site to another. (This propagation delay is about 0.1 sec. for the ARPANET.) The other problem is that communication channels and computer sites are susceptible to failures, giving rise to networks that may have constantly changing topologies.

1.3 Key Technical Problems

It is noted that some of the problems associated with distributed databases are the same as those for the non-distributed (centralized) database and can therefore use the same solutions. Such problems include*: choosing a good data model, designing a schema, etc. However, mainly because of the two implementation problems associated with the distributed database, the following problems require significantly different approaches:

(1) data retrieval (or query processing) - a query accessing data stored at different sites requires that data must be moved around in the network. The communication delay, and hence the response time, depends strongly on the choice of a particular data storage and transfer strategy.

(2) update synchronization (or concurrency control) - in centralized databases, locking is the standard method used to maintain consistency

---

* The reader is referred to [DATE77] for a definition of these terms.

among redundant copies of data. The distributed nature of the data in DDB means that setting locks produces long message delays.

(3) reliability/survivability - the network introduces new components (communication links, computers) where failure can occur, and hence the associated problems of failure detection and failure recovery.

(4) physical database design (file allocation) - the problem of how many copies of each data file to maintain and where to locate them. The use of additional redundant copies generally means reduced communication delay associated with data retrieval. Unfortunately, it also means increased delay associated with update synchronization. The problem is difficult not only because of varying file request rates due to the users, but also because of the dynamic nature of the network topology. Nodes may be lost due to computer and communication link failures or a particular node moving out of range of the communication medium. The topology, and hence the associated file allocation, changes again when new nodes are added to the network due to computer and link recovery.

1.4 Performance Modelling

1.4.1 Need for a Performance Model

Our literature research has led us to conclude that a good model of a distributed database system is essential to research in this area. Such a model will enable us to compare the performance of the different algorithms proposed. It can also help identify the decisive factors of system performance and consequently determine the direction one should take in further research on improvement.

Such models should be flexible enough to allow for the testing and comparison of alternative synchronization and query processing algorithms.

One possible approach to performance modeling is to develop a simulation model. However, simulation models are generally expensive. Besides, a simulation model only gives results for one particular configuration of the problem, as defined by a specific set of parameters, and does not provide as much insight as analytic models on the relationship between the results and the parameters. We shall therefore concentrate on developing an analytic model.

1.4.2 Review of Research

While the literature on DDB abounds in concurrency control and query processing algorithms, there is very little work done on comparing the performance of the different proposals. Bernstein and Goodman [BG80] analyzed the performance of principal concurrency control methods in qualitative terms. The analysis considers four cost factors: communication overhead, local processing overhead, transaction restarts and transaction blocking. The assumption is that the dominant cost component is number of messages. Thus distance between database sites, topology of network and queueing effects are completely ignored. A quantitative comparison is described in the thesis of Garcia-Molina [GARC79]. He compared several variants of the centralized locking algorithm with Thomas's Distributed Voting Algorithm [THOM79] and the Ring Algorithm of Ellis [ELLI77]. The major assumptions are (1) a fully redundant database, and (2) the transmission delay between each pair of sites is

constant. The first assumption requires that the whole database is fully replicated at each node. This is necessary because Garcia-Molina did not want to model query processing, which would have been necessary for a general (not fully redundant) database . The second assumption means that the topology, message volume and queueing effects of the communication subnetwork are ignored.

1.5 Outline of Thesis

The goal of this research is to develop a performance model of a DDB. In particular, the model will be used to compare the performance of different concurrency control algorithms. In Chapter 2 we describe some well-known concurrency control algorithms. In chapter 3, we develop the communication subnetwork model, which is an important component of our DDB model, described in Chapter 4. Chapter 5 starts with a review of existing query processing algorithms. We have found them hard to analyze and have developed two new query processing algorithms that are easy to analyze. In Chapter 6, we introduce the conflict model, which allows us to calculate such important parameters as the probability of conflict and the delay due to conflicts. Chapter 7 consists of four numerical examples. They are based on the same communication subnetwork so that comparisons can be made between different concurrency control algorithms. We conclude in Chapter 8 with a discussion of our results and suggestions for further research.

CHAPTER 2

CONCURRENCY CONTROL

This chapter is devoted to concurrency control, with two objectives: to define the notion of correctness of a concurrency control method and to describe existing concurrency control algorithms. Although the literature abounds in concurrency control methods, they can be divided into two major approaches, namely, two-phase locking and timestamp ordering. These will be discussed in sections 2.3 and 2.4. First we shall describe the concurrency control problem.

2.1 The Concurrency Control Problem

In a database system, data items are related in certain ways. These relationships can be thought of as assertions about the data items. Consider two data items x and y; examples of assertions are: $x = y$, $x > 0$. The database system is said to be consistent when its data items satisfy all its assertions, or consistency constraints.

Assume that the basic unit of user computation is the transaction. A transaction executes in three steps, each of which is assumed indivisible:

(1) Read - the transaction reads some data into a local workspace.

(2) Computation is performed on the workspace

(3) Write - some values in the workspace are written back into the database.

If user requests are not coordinated, the execution of steps in different transactions from different users may be interleaved in any order. Assume that each transaction preserves the consistency of the database when executed alone. The execution of many interleaved trans-

actions may bring a consistent database into an inconsistent one (see [EGLT76], [GRAY78],[BG80]). For example, suppose the present value of x is 10 and two transactions T_1 and T_2 execute the following program: increase x by 1. If T_1 and T_2 are executed one after another, the new value of x is 12. However, if they are executed in the following interleaved order:

(1) T_1 reads x = 10

(2) T_2 reads x = 10

(3) T_1 increases x by 1, x = 11

(4) T_2 increases x by 1, x = 11

then the new value of x is 11 which is incorrect. Other examples of concurrency anamolies can be found in [EGLT76], [GRAY78]. It is the task of the concurrency control mechanism of the database system to safeguard database consistency.

2.2 Serializability

The notion of correctness in this thesis is that of serializability. A set of transactions executes serially if each transaction executes its write step before the next transaction executes its read step. That is, the transactions are not interleaved. A serial execution of transactions preserves database consistency since each transaction, by assumption, maps the original consistent database to another consistent database. A sequence of interleaved transactions is serializable if it produces the same final state as a serial execution of those same transactions. Since a serial execution preserves consistency, a serializable execution also preserves consistency.

Serializability is an appealing correctness criteria, since it can be guaranteed without having the database system know the precise computation performed by each transaction. It needs only know the portions of the database read and written by each transactions. (see, e.g.[BSW79], [KP79]). Other criteria of correctness bave been proposed (see, e.g. [GLPT75]), but serializability is the most popular among researchers in the area. ([EGLT76],[RSL78],[BSW79], etc.)

2.3 Two-Phase Locking

2.3.1 Specification

In two-phase locking, every data item has a lock associated with it. At any time, no more than one transaction can hold the lock on a data item. If a transaction requests locks on all needed data items before starting and releases the locks after completion,serializability is preserved. "Two-phase" refers to the requirement that locks will be obtained in two phases, a growing phase and a shrinking phase. It is important that once a transaction has released a lock, it will not obtain any more locks. The point in time at the end of the growing phase is called the locked point[BSW79]. The serializability of two-phase locking is induced by the locked points. Several authors have proved that two-phase locking guarantees serializability([BSW79],[EGLT76]).

One major drawback of two-phase locking algorithms is the possibility of deadlocks. Deadlocks occur when two transactions are waiting for each other to release locks on some data items. (see section 2.4). There are different versions of two-phase locking algorithms. The two major methods are described in the next sections. The solution of the deadlock

problem is quite different for different methods.

2.3.2 Distributed Two-Phase Locking

Locks on data items are managed by a scheduler or lock manager. In the basic implementation, or Distributed Two-Phase Locking, the schedulers are distributed with the data. At each database site, there is a scheduler, responsible for the data items at that site.

A transaction reading a data item X can send a read request to any site containing a copy of X and request the lock on X from the scheduler at that site. On the other hand, a transaction updating a data item has to send write messages to all sites having a copy of X and request locks from the schedulers at all of those sites*.

This approach is more efficient than the Centralized Two-Phase Locking Algorithm (described in the next section) in that there is no central node to serve as a potential bottleneck since the schedulers are distributed with the data. However, the distribution of the schedulers means that deadlock detection is infeasible. For example, transaction A may be waiting for Transaction B to release its locks at node 1, while transaction B may be waiting for Transaction A at node 2. There are no deadlocks locally at nodes 1 and 2. However, the system has a deadlock and will not be able to detect it unless locking information is communicated from node to node. This communication will put a heavy burden on the communication subnetwork. Therefore deadlock detection is infeasible, and more complex solutions to the deadlock problem must be used.

---

*One way of handling redundant data is to assume that redundant copies of a data item X are distinct data items X_1, X_2,... X_n. Reads can be processed at any of the copies, while writes must be implemented at all copies.

2.3.3 Centralized Two-Phase Locking

There are many variations on the basic implementation, one of which is Centralized Locking[AD76]. In this case, the scheduler is located at one site, the central node. Before accessing data at any site, locks must be obtained at the Central Node.

This approach has the advantage of central control and relatively easy deadlock resolution (which will be discussed in the next section). However, the creation of the central node means that all transactions have to go through this node, thereby producing a potential bottleneck. Besides, when the central node fails, the system cannot function anymore. Various remedies are proposed, including the use of multiple central nodes to solve the bottleneck problem and the use of redundant central nodes to solve the reliability problem.

2.4 Deadlocks

In two-phase locking, a transaction is asked to wait when the data item it accesses is locked by some other transactions. If this waiting is uncontrolled, a deadlock may result. For example, if transaction A is waiting for transaction B to release its locks while transaction B is waiting for transaction A to release its locks, then neither transaction can complete because they cannot get all the data items required, neither transaction releases its locks and there is a deadlock. A deadlock is best illustrated by a waits-for graph[KC74], which is constructed as follows: For all pairs of transactions A and B, an edge is drawn from transaction A to transaction B if A is waiting for a lock currently owned by B. There will be a deadlock in the system iff the waits-for graph contains a cycle. To solve the deadlock problem, we can employ either deadlock detection or deadlock prevention.

2.4.1 Deadlock Detection

In deadlock detection, transactions are allowed to wait for locked items in an uncontrolled manner. Periodically, a deadlock detector constructs the waits-for graph of the system and determines whether there are any deadlocks by searching for cycles in that graph. If a cycle is found, one of the transactions in the cycle is restarted. Hopefully, enough information will be available to allow one to choose the cycle-breaking transaction intelligently, so that the amount of resources wasted is minimum. Unless all locking information is available at one node, as in the Centralized Locking Algorithm, deadlock detection is not practical because of the communication of locking information that is necessary.

2.4.2 Deadlock Prevention

In deadlock prevention, transactions are allowed to wait for locked items in a controlled manner to eliminate the possibility of deadlocks.

Ordered Queues

One way to prevent deadlocks is to require that all transactions request locks in some universally specified order, i.e. wait for X first, then Y, etc. This has the special property that transactions never have to be restarted.

Prioritized Transactions

Another method to prevent deadlocks is to assign priority to each transaction and to require that when transaction conflict,only higher priority transactions can wait for lower priority transactions, or vice versa. Consider the waits-for graph of such a system. Every edge in the

graph is in priority order, i.e. $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_i$. A cycle is a path from a node to itself and since it is impossible to have a transaction which has lower priority than itself, no cycles can occur, and the system is deadlock free.

One convenient way to assign priority is to use timestamps, the older the timestamp the higher the priority. Two methods are proposed by Rosenkrantz, et. al.[RSL78]. Suppose T_i tries to wait for T_j:

(1) Wait-die System

If T_i has higher priority, it is allowed to wait; if T_i has lower priority, it is restarted.

When T_i is restarted, it releases all resources that it has previously locked. The database management system then resubmits lock requests for T_i. To prevent cyclic restarts, i.e. being restarted over and over again, T_i retains its original timestamp.

(2) Wound-wait System

If T_i has higher priority, T_j is wounded and T_i waits until the wound takes effect; if T_i has lower priority, it is allowed to wait.

When a transaction T_j is wounded, it sends wound messages to all sites where T_j is being processed. If the transaction has not initiated termination, i.e. the write phase of the two-phase commit*, then the transaction is restarted. Otherwise, the transaction is allowed to finish because in this case there is no danger of deadlock.

---

* See section 4.3 for a description of the two-phase commit algorithm.

In the wait-die system, younger transactions are restarted when they conflict with older transactions, while in the wound-wait system, the younger transactions are allowed to wait. Younger transactions arrive into the system later than older transactions, and hence are likely to arrive at a given node later too. Therefore, the majority of conflicts in the system is of the type: younger transactions waiting for older ones. Hence, under wait-die, most transactions will be restarted, while under wound-wait, most transactions are allowed to wait. Since waiting presumable consumes less resources than restarting, wound-wait seems more efficient than wait-die.

2.5 Timestamp Ordering

2.5.1 Specification

In timestamp ordering, each transaction is assinged a timestamp. This timestamp is guranteed to be globally unique by ensuring that no new timestamp will be assigned at the same site until the clock has ticked, and by appending the site number to the low order bits of the timestamp. The timestamp is attached to all read and write operations issued on behalf of that transaction. Each data item has a timestamp equal to that of the last write operation. The database management system is required to process the transactions in timestamp order. This is accomplished by transaction restarts and transaction blocking. Suppose a transaction sends a RR (request to read message) with timestamp TS_1 to site α to read data item X (with timestamp TS_2). If $TS_1 < TS_2$, the transaction must be restarted and resubmitted with a new (and bigger) timestamp. If $TS_1 < TS_2$, the RR must wait until all write operations with timestamps less than TS_1 have been

processed. Therefore, the RR must wait for the arrival of all write operation (with timestamp TS_3) such that $TS_3 > TS_2$. Since this write operation may never arrive, the wait may be infinite. To alleviate this problem, the database system has to generate periodically null-write messages, i.e. write messages with only a timestamp but no new value for the data item, from each site.

It can be shown that timestamp ordering guarantees serializability [BG80] and the serialization order is the timestamp order.

2.5.2 SDD-1

SDD-1 ([BG80], [BGR80]) is a specialized version of a timestamp ordering (T/O) synchronization algorithm. The basic T/O algorithm seeks to guarantee serializability by re-ordering transactions so that they will be processed in timestamp order. The algorithm is applied to all transactions, regardless of whether conflicts do indeed exist. SDD-1 tries to improve on the basic T/O algorithm by incorporating two new features:

(1) transaction classes - a transaction class is defined by its readset and its writeset. Conflicts between transactions can be determined by conflicts between their respective classes.

(2) conflict graph - conflicts between classes can be analyzed during database design. It is determined that four protocols (P1,P2,P3,P4) with varying costs of synchronization, are sufficient to guarantee serializability.

In defining transaction classes, transactions that arrive at different TM's will be designated as different transaction classes, even if their readsets and writesets are the same. Moreover, SDD-1 assumes that messages between

each pair of sites will be sent and arrive at their destinations in timestamp order.

The basic architecture of SDD-1 is shown in Fig. 2.1. It consists of Transaction Modules (TM's) and Data Modules (DM's). The TM supervises transaction executions while the DM manages the data at the local site. TM's and DM's may be located together at the same computer site or reside at different sites.

The algorithm is best illustrated by an example (Fig. 2.2). Suppose a transaction i arrives at TM_α. To process this transaction, TM_α follows the following steps:

(1) Look in class table and find the class of i, say $\bar{i}$.

(2) Determine the conflicting transaction classes $\bar{j}$, $\bar{k}$, etc., and the required synchronization protocols.

(3) Query Processing : TM_α devises a query processing strategy to read data at the DM's and produce the result at TM_α. Suppose the strategy requires reading a data item at DM_α, then TM_α sends a RR (request to read) message to DM_α with a Read Condition $<TS_i, (\bar{j}, \bar{k}, \ldots)>$. The read condition identifies the conflicting classes $\bar{j}$, $\bar{k}$, etc. and the timestamp of i (TS_i). When the read message arrives at DM_α, the synchronization protocol dictates that it must be processed after all write messages from conflicting classes (i.e. $\bar{j}$, $\bar{k}$, etc.) with timestamps less than TS_i, but before those write messages with timestamps greater than TS_i. Hence, if the read message (with timestamp TS_i) arrives at DM_α later than a write message (with timestamp TS_w) generated from another TM, where $TS_w > TS_i$, then the read message cannot satisfy the required

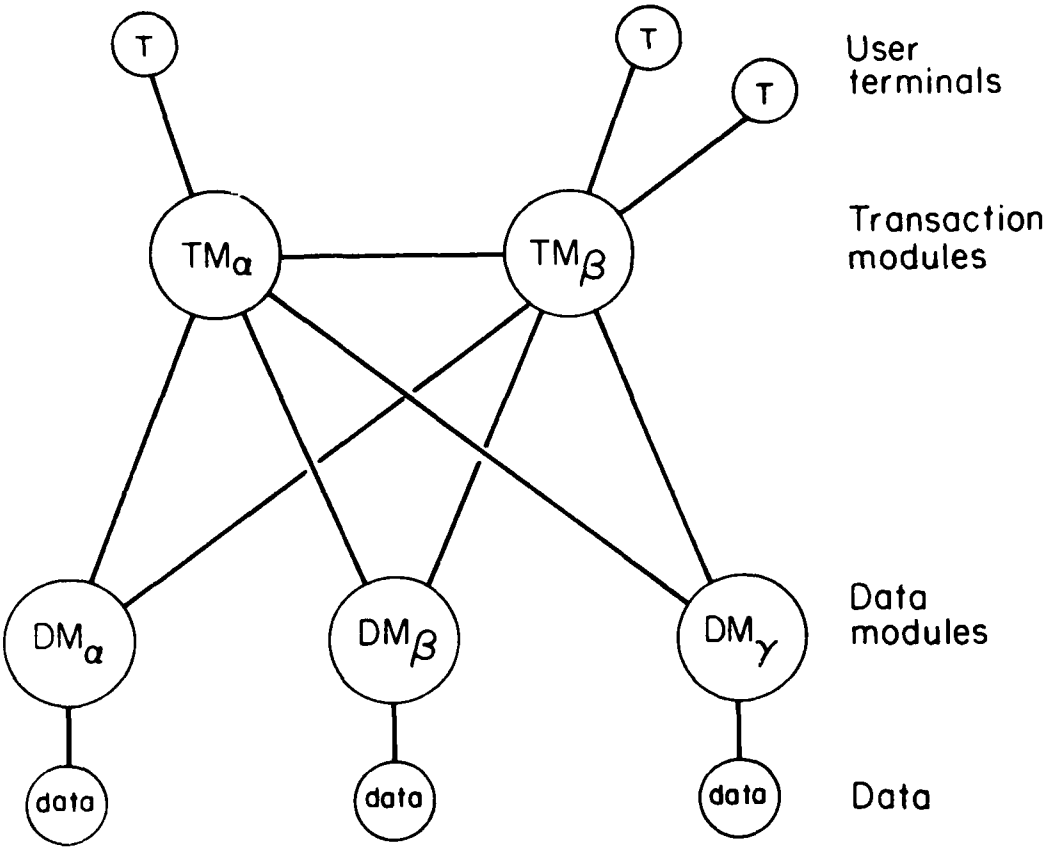
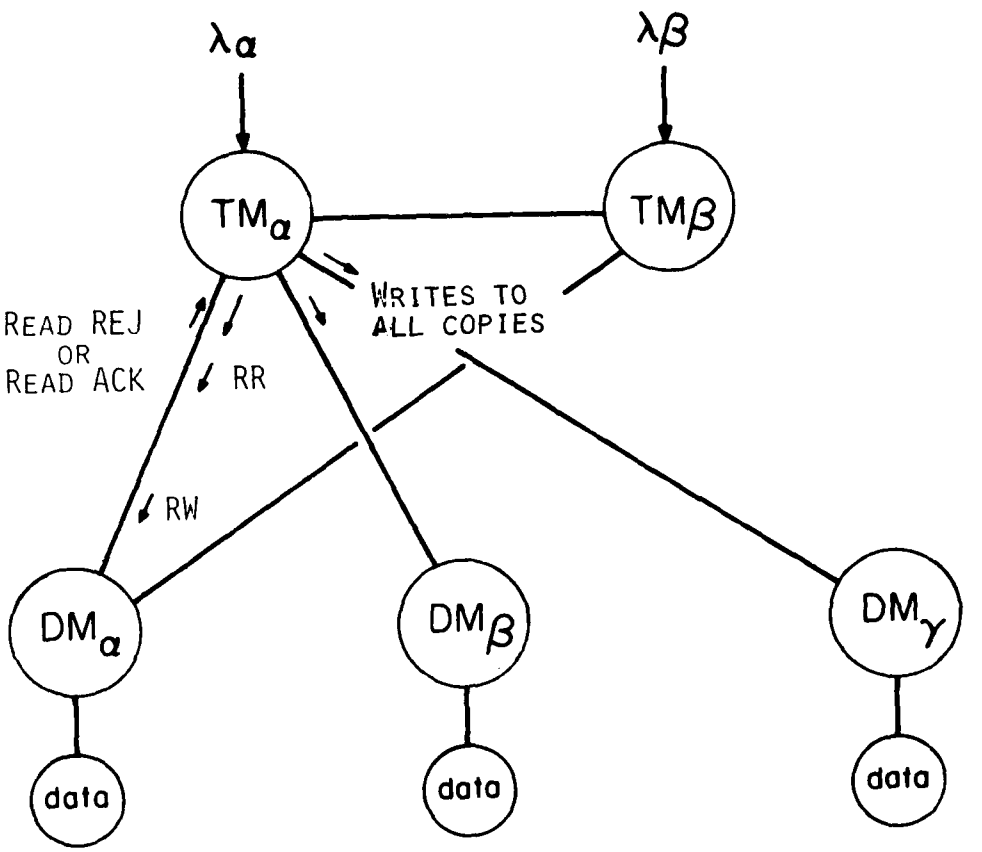


Figure 2.1 Basic Architecture of SDD-1



RR - REQUEST TO READ FROM DM ONTO LOCAL WORKSPACE
RW - REQUEST TO WRITE FROM WORKSPACE BACK INTO DM

Figure 2.2 A Simple Example of SDD-1

synchronization protocol and must be rejected and transaction i restarted by TM_α with a later timestamp. Note that any other read messages on behalf of transaction i have to be resubmitted with this new timestamp. The scenario described above, namely messages arriving at DM_α in reverse order to their timestamps, is possible because of different transmission delays on different communication channels. If the read message is not rejected, then it must wait until its read condition is satisfied, i.e. it must wait for the arrival of a write message or a nullwrite message with timestamp greater than TS_i from all conflicting transaction classes. These write messages must then wait for the read to take place before they can be implemented.

(4) Write: TM_α sends RW (request to write) messages to all DM's containing data items that have to be updated. A RW consists of the new value of the data item and a timestamp equal to that of transaction i. All write messages can be implemented as soon as they are received at the DM's. The rule is as follows: If the timestamp of the RW is smaller than that of the stored data item it tries to write, the RW is ignored. Otherwise, the value and timestamp of the stored data item will be updated, after pending reads, if any,are processed. This is known as the Thomas Write Rule [THOM79]. Nullwrites are treated differently from RW's in that they do not update the timestamp of the data item.

2.6 Other Concurrency Control Algorithms

Our discussion of concurrency control algorithms will not be complete wihout at least a brief descripiton of Thomas' Majority Consensus Algorithm [THOM79] and Ellis' Ring Algorithm [ELLI77]. These two algorithms are

among the earliest algorithms proposed for concurrency control. However, they are devised for fully redundant databases and can be shown to be rather inefficient ([BG80]). Therefore, we shall not attempt to model them.

2.6.1 Thomas' Majority Consensus Algorithm

Thomas devised an algorithm for fully redundant databases. Reads are processed at the site where the transaction originates. A proposed write, or update is passed from site to site. Each site will vote yes, no or pass on the update. When a majority of sites have voted yes, the update is installed. Voting has therefore effectively replaced locking. The locked point, using two-phase lock terminology, is reached when the site originating the update has received a majority of yes votes.

This algorithm is impractical since it assumes a fully redundant database. Even for fully redundant databases, it is inefficient since conflicts between transactions must be solved by restarts in contrast to locking and timestamp ordering algorithms which can also resolve conflicts by transaction blocking (i.e. waiting for locks to be released or read conditions to be satisfied.)

2.6.2 Ellis' Ring Algorithm

In Ellis' Algorithm, the database is fully redundant and the communication subnetwork is configured as a ring. To effect an update, a transaction moves successively from site to site on the ring, obtaining a lock on the entire database at each site. This means that all transactions must be executed serially, and no parallel processing is possible. Hence, the algorithm is inefficient.

CHAPTER 3

COMMUNICATION SUBNETWORK MODEL

Since the distributed database is managed on a communication network, and we can think of transactions (i.e., queries and updates) as competing among themselves for the available resources of the network, it seems natural to model the communication network by a network of queues. We have attempted to employ Jackson's Queueing Networks since, provided Jackson's assumptions are satisfied, the resulting model is very powerful. However, we have found that Jackson's model is inadequate for our purposes. In this chapter, we introduce Jackson's model, describe how it can be used to model a communication subnetwork and point out its inadequacies. We then propose a new "independent queues" model of a communication subnetwork. This model is adopted in this thesis for the analysis of communication delay and is described in section 3.3

3.1 Jackson's Network of Queues

3.1.1 Basic Model

The model introduced in Jackson[JACK57] is an arbitrary network of queues, consisting of N nodes where the ith node consists of m_i exponential servers, a first-come-first-served queue discipline and unlimited queueing capacity. The external input stream to node i is Poisson with rate γ_i, and these external input streams are assumed to be independent. The service times at node i are independent and have a common exponential distribution with parameter μ_i, and are also independent of the customer arrivals at node i. A customer leaving node i is immediately and independently routed to node j with probability

p_{ij}, and he departs the system with probability $q_i = 1 - \sum_{j=1}^{N} p_{ij}$. If we denote the total arrival rate (including external and internal arrivals) to node i by λ_i, we see that the following equations must be satisfied:

$$\lambda_i = \gamma_i + \sum_{j=1}^{N} \lambda_j p_{ji} \qquad i = 1,2,\ldots N \qquad (3.1)$$

Let the state variables for this N-node system consist of the vector $(k_1,k_2,\ldots k_N)$, where k_i is the number of customers (including those in service) at the ith node, and the equilibrium probability associated with this state be denoted by $p(k_1,k_2,\ldots k_N)$. Similarly, let the marginal probability of finding k_i customers at the ith node be $p_i(k_i)$. Jackson showed that, provided the utilization is less than one at each node, i.e.,

$$\rho_i = \lambda_i/(m_i\mu_i) < 1 \qquad i = 1,2,\ldots N$$

then the joint distribution for all nodes factor into the product of each of the marginal distributions, i.e.,

$$p(k_1,k_2,\ldots k_N) = p_1(k_1)p_2(k_2)\ldots p_N(k_N)$$

and $p_i(k_i)$ is given as the solution of the classical M/M/m system, i.e.,

$$p_i(k_i) = \begin{cases} p_i(0)(m_i\rho_i)^{k_i}/k_i! & 0 \le k_i \le m_i \\ p_i(0)(\rho_i)^{k_i}(m_i)^{m_i}/m_i! & m_i < k_i < \infty \end{cases} \qquad (3.2)$$

This says that whenever an equilibrium condition exists, each node in the network behaves as if it were an independent M/M/m queue with Poisson input, although in general the total input (including internal transfers) at each node is not Poisson.

3.1. General Model

Since Jackson first published his basic model in 1957, there have been numerous extensions proposed. The most general results have been derived by Baskett et al.[BCMP75]. Their model consists of N nodes and L classes of customers. Customers travel through the network according to transition probabilities

$$P_{i,r;j,s} = P\ (\text{next node is } j, \text{ next class is } s \mid \text{current node is } i \text{ and current customer class is } r)$$

where i, j = 1,...N; r, s = 1,...L.

The system can be closed for certain classes of customers and open for others. If the system is closed for customers of class r, then the number of such customers within the system is fixed at a constant number k(r). In an open network, the total arrival rate to the network is Poisson with mean rate dependent on the total number of customers in the network. An arrival enters node i in class r with a fixed probability (state independent) of q_{ir}.

There are also four kinds of service nodes:

Type 1 Node : The service discipline is first-come-first-served (FCFS) with a single server; all customers have the same, negative exponential service time distribution with rate $\mu(j)$ where j is the number of customers at the node.

Type 2 Node: There is a single server at the node, the service discipline is processor sharing (i.e. when there are n customers in the node, each is receiving service at a rate of 1/n times total service rate), and each class of customer may have a distinct service time distribution. The service time distributions have rational Laplace transforms.

Type 3 Node: The number of servers in the service center is greater than or equal to the maximum number of customers, and each class of customer may have a distinct service time distribution.. The service time distribution have rational Laplace transforms.

Type 4 Node: The queueing discipline is preemptive-resume last-come-first-served(LCFS) with a single server. Each customer class may have a distinct service time distribution which must have a rational Laplace transform.

Note that any distribution can be approximated by a distribution with rational Laplace transforms.[KLEI75].

The traffic equations for this general Jackson Model are:

$$e_{is} = \gamma_{is} + \sum_{i,r} e_{ir} p_{i,r;i,s} \qquad \begin{matrix} i = 1,\ldots N \\ s = 1,\ldots L \end{matrix} \qquad (3.3)$$

where e_{is} is the arrival rate of class s customers to the ith node.

The states of the system involve a rather complex description [illegible] customers, their class and their stage of attained [illegible]. Baskett et al.[BCMP75] proved that the equilibrium probabilities are given by

$$\pi(x_1, x_2, \ldots, x_N) = \pi_1(x_1)\pi_2(x_2)\ldots\pi_N(x_N)$$

where the state vector $x_1,\ldots,x_N$ consists of components x_i (which are [illegible] that represent the conditions prevailing at node i. [illegible] the node state x_i is a [illegible] [illegible] the total number of customers at the [illegible] that describes the type of node, [illegible] the [illegible] specifies the system state at [illegible] where [illegible] the number of customers

of class r in service center i. Let n_i be the total number of customers at service center i and let $1/\mu_{ir}$ be the mean service time of a class r customer at service center i, the equilibrium probabilities are given by

$$P(S = (y_1, y_2, \ldots y_N)) = Cd(S)\ q_1(y_1) \ldots q_N(y_N) \qquad (3.4)$$

where $q_i(y_i) = n_i! \{ \sum_{r=1}^{L} (1/n_{ir}!)(e_{ir})^{n_{ir}} \} (1/\mu_i)^{n_i}$ for Type 1 nodes,

$$q_i(y_i) = n_i! \prod_{r=1}^{L} (1/n_{ir}!)(e_{ir}/\mu_{ir})^{n_{ir}} \quad \text{for Type 2 or 4 nodes,}$$

$$q_i(y_i) = \sum_{r=1}^{L} (1/n_{ir}!)(e_{ir}/\mu_{ir})^{n_{ir}} \quad \text{for Type 3 nodes.}$$

This result is remarkable not only because of the product form exhibited, but also because of the fact that general service time distribution(with rational Laplace transforms) for the different classes of customers yield the same result as exponential service time distributions, since only the mean service rate is included in the results.(See Equation 3.4)

Further simplification, by aggregating over all classes, reveals that for node type 1, 2 and 4 the equilibrium distribution for the number of customers at each node is the same as the distribution in an M/M/1 system while for node type 3, the corresponding distribution is that of an M/M/∞ system.

Extension to Models of Computer Communication Networks

We will now extend the model to a more general network such as the [illegible] consisting of different types of processors with different service

3.2.1 Model Description

Suppose we want to model a communication subnetwork by Jackson's Basic Model. The modeling process can best be illustrated by an example.

Consider the simple network shown in Figure 3.1, which contains three computer sites connected by directed communication links. Messages (which can be update, query, file transfer, or acknowledgement messages) enter the different nodes destined for other nodes. The quantity γ_{ij} is the rate of arrival of messages at node i destined for node j. C_{ij} is the capacity of the channel from i to j. If we assume that the message arrival is Poisson, that the message lengths are exponentially distributed with mean $1/\mu$, and that the independence assumption holds, i.e., each time a message enters a new channel, a new length is chosen independently from the exponential pdf, we can then model this network by the basic Jackson's model. Each channel corresponds to a node in Jackson's model*. It is further assumed that each channel serves messages (customers) in a FCFS queue discipline with unlimited queueing capacity and an exponential service time $1/C_i$ where C_i is the capacity of the channel. A message leaving channel i is immediately routed to channel j with probability p_{ij}, and departs the system with probability q_i. Figure 3.2 shows this simple three-computer-site example again (with the paths taken by the different messages indicated by letter lines) and its equivalent Jackson's model [illegible] node for channel C_{12}: both γ_{12} and γ_{13} messages use this channel, but once they reach computer site 2, the γ_{13} messages, correspond-

* [illegible] to model store-and-forward communication centers with [illegible] a misleading interpretation. We shall [illegible] what nodes and what the servers. In Jackson's model, [illegible] network, it is the channels.

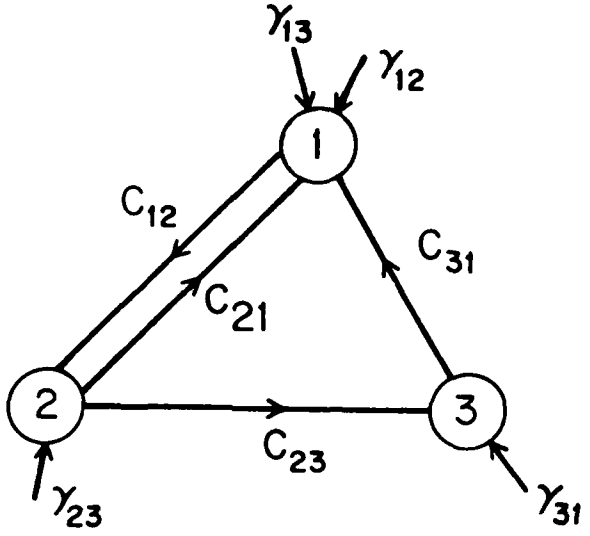


Figure 3.1 A Three-Node Communication Network

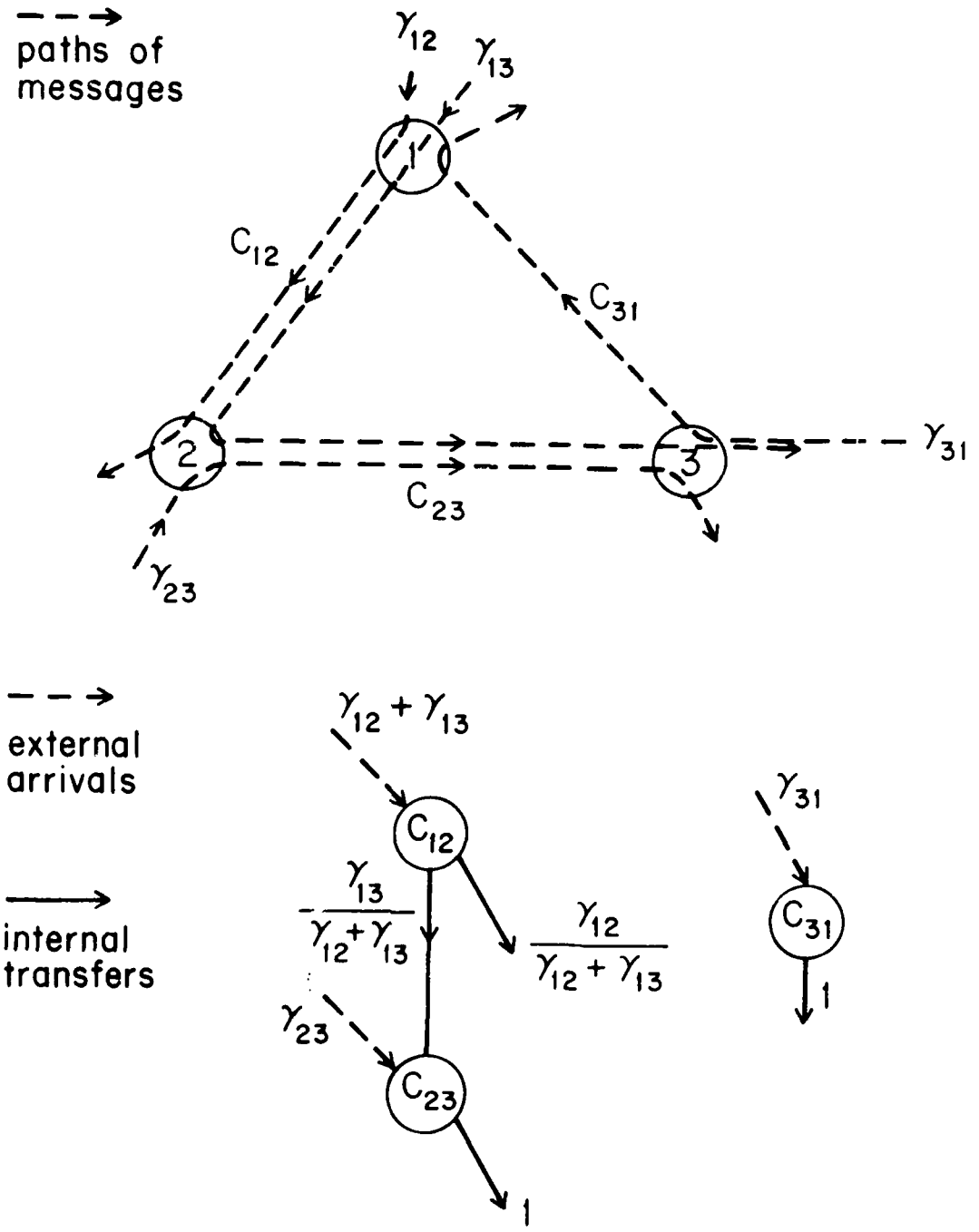


Figure 3.2 Communication Network and Equivalent Jackson's Model

ing to a fraction $\gamma_{12}/(\gamma_{12} + \gamma_{13})$ of the total messages passing through channel C_{12}, leaves the system while the rest is transferred to channel C_{23}. Similarly, we can find the routing probabilities for the other channels.

3.2.2 Assumptions

It is noted that in order to use the basic Jackson's model, it is necessary to make the following assumptions, which approximate reality with varying degrees of success:

(1) Poisson message arrivals

(2) One class of message with exponentially distributed message lengths

(3) Stationarity and Independence of the Stochastic Processes (1) and (2)

(4) First -come-first-serve queueing discipline

(5) Independence of service times at different communication channels.

(6) Random routing of messages, irrespective of message destinations.

(7) Unlimited queueing capacity at each node.

(8) Noiseless channels and perfectly reliable nodes and channels.

How well does Jackson's model approximate the communication subnetwork of interest ?

The author is not aware of any empirical studies to justify assumptions (1) to (3) for a communication network. However, certain data obtained by Molina [MOLI27] for telephone traffic correspond well to these assumptions.

Assumption (2) is unrealistic. Since we are interested in the communication subnetwork of a DDB, there will be different classes of messages with different lengths. At the very least, we would like to distinguish between long messages such as file transfers and short messages such as lock requests and lock grant messages. The basic Jackson's model only allows one class of message.

The general Jackson's model does allow different classes of messages for queueing disciplines other than FCFS. However, in a store-and-forward network, the queueing discipline is best described as FCFS.

Assumption (4) is relaxed in the general Jackson's model, and other queueing disciplines as described in section 3.1.2 are allowed. In particular, one can model statistical multiplexing by Type 2 service node (processor sharing) and some preemption strategies by Type 4 service node.

Assumption (5) was first introduced by Kleinrock [KLEI64]. In Jackson's model the service time at each node is an independent random variable, while in Kleinrock's model, as in our communication network model, the service time for a given message on channel i depends on the message length b and the capacity C_i of the channel, i.e. b/C_i. The service times at different channels are therefore not independent. Besides, there is a dependency between the interarrival times and lengths of adjacent messages as they travel within the network. (See, for example, Equation (3.4) in Kleinrock [KLEI64]). To alleviate this problem Kleinrock introduced the Independence Assumption:

Each time a message is received at a node within a network, a new length b is chosen independently from the pdf:

$$p_b(b_0) = \mu e^{-\mu b_0} \qquad b_0 \geq 0$$

Kleinrock has treated this assumption extensively in [KLEI64] which includes a computer simulation model and argued that so long as there are multiple channels coming into and leaving a communication node, this assumption is reasonable.

Assumption (6) says that once a message has finished transmission at channel i, it will be routed to channel j with a certain probability p_{ij}, irrespective of its distination. This assumption is invalid for most networks we are interested in, and it is another reason why we believe Jackson's model is inadequate. Consider, for example, the ARPANET, where messages are routed in the network over different channels according to their destinations.

Assumption (7) is unrealistic for networks with limited buffer space such as the ARPANET. However, with the development of cheaper and smaller core memories, buffer space can be practically unlimited in the near future.

Assumption (8) is realistic if one assumes that there is a layer of software, superimposed on the communication subnetwork, which is responsible for detecting errors in transmission (e.g. using the cyclic redundancy check) and for retransmission of noisy messages. Of course, in this case the service time has to be adjusted to reflect the increased volume of traffic due to retransmissions.

It is mainly because of Assumptions (2), (5) and (6) that we have decided that Jackson's model is inadequate in modelling a communication subnetwork. In the next section, we shall propose a new Independent Queues Model that will alleviate these difficulties.

3.3 Independent Queues Model of Communication Subnetwork

3.3.1 Model Description

The model is based on the Independent Queues Assumption*:

---

*This assumption is suggested by Prof. Robert Gallager.

Consider a communication subnetwork with N channels. Suppose the total arrival rate of messages to channel i is λ_i. λ_i includes both the external arrival rate of messages at channel i which is Poisson and the internal transfer from other channels to channel i according to some routing strategies. Let $p(k_1,k_2,\ldots k_N)$ be the equilibrium joint pmf of the number of messages at the different channels, and $p_i(k_i)$ be the equilibrium marginal pmf of the number of messages at channel i under Poisson input rate λ_i, then, provided the utilization at each channel is less than one, it is assumed that

$$p(k_1,k_2,\ldots k_N) = p_1(k_1)p(k_2)\ldots p_N(k_N)$$

In particular, if the message lengths are exponential, the Independent Queues Assumption says that, in equilibrium, channel i behaves as if it were an independent M/M/1 queue with Poisson input rate λ_i.

The Independence Queues Model allows us to calculate the message delay for the network very easily. The average message delay T_i on channel i is $1/(\mu C_i-\lambda_i)$* where C_i is the capacity of channel i, λ_i is the total arrival rate of messages to channel i, and $1/\mu$ is the average length of messages.

Consider the average number of messages (both in queue and in service) in channel i in equilibrium, n_i. Applying Little's Formula [LITT61] , this is just $\lambda_i T_i$. Applying Little's Formula again for the whole network and denote the average number of messages in the whole network by n, we have

$$\gamma T = n = \sum_i n_i = \sum_i \lambda_i T_i$$

---

*Average service time for M/M/1 queue with arrival rate λ_i and service rate μC_i.

where γ = sum of external arrival rates to network, T = average message delay for the whole network, then

$$T = \frac{1}{\gamma} \sum_i \frac{\lambda_i}{\mu C_i - \lambda_i} \tag{3.5}$$

Note that the derivation of Equ.(3.5) does not require the service times at the different channels to be independent. The only requirement is that the service time at each channel i can be approximated as a function of the input traffic λ_i.

Our model ignores the propagation delay for the energy of a bit to travel from one end of a channel to the other. Even though the speed of propagation is a significant fraction of the speed of light, the propagation time may be significant if the path is long. In addition, there is the additional delay introduced during local processing at each of the channels. Let P_i, K_i be the propagation and local processing delays associated with the ith channel, then our average message delay given above must be rewritten as:

$$T = \frac{1}{\gamma} \sum_i \lambda_i \left(P_i + K_i + \frac{1}{\mu C_i - \lambda_i} \right) \tag{3.6}$$

3.3.2 Assumptions

The Independent Queues Model of a communication subnetwork requires the following assumptions:

(1) Poisson message arrivals.

(2) First-come-first-served queueing discipline.

(3) Independent Queues Assumption.

(4) Unlimited queueing capacity at each node.

(5) Noiseless channels and perfectly reliable nodes and channels.

Assumptions (1),(2),(4) and (5) are also necessary for Jackson's model. These assumptions have been discussed in section 3.2.2 and will not be repeated here. Assumption (3), like Kleinrock's Independence Assumption, is hard to justify. The rationale for this assumption is that it makes the problem analytically tractable and seems to give reasonable results. While this assumption seems to be stronger than the Independence Assumption, it affords us more flexibility. For example, we can now model general message length distributions. Instead of independent M/M/1 queues, we then obtain independent M/G/1 queues. We can also model different classes of messages using the M/G/1 queues. In addition, we can model more accurately the routing of messages in the network.

3.4 End-to-end Transmission Delay

A key parameter in our DDB model is the end-to-end delay which is the elapsed time from the transmission of a message at its source to the delivery of the message at its destination.

Since messages typically preserve their length as they traverse the system, the interarrival and service sequences at each communication channel, and the service times at successive communication channels are dependent. The distribution of the end-to-end delay is thus mathematically intractable. A number of results concerning this delay are presented by Calo in [CALO80]. These include : ordering relations for the successive waiting times in the channel; waiting time properties under extreme conditions; and simple bounds for systems with uniformly bounded service processes. However, these results are not useful to us

in the DDB model. In fact, even if we make the Independent Queues Assumption, the distribution of end-to-end delay still eludes us. Recall that if we make the Independent Queues Assumption, the number of messages, and hence the service time, at all channels are independent in equilibrium. This means that at a particular time, say t, the service times at all channels are independent. However, when a message is traversing the system from one channel to another, it will be receiving service at successive channels at different times. The service time of a particular message at successive channels are thus dependent. It is this dependence that makes the analysis difficult.

In order to analyze the end-to-end delay, approximations are made.

3.4.1 Exponential End-to-End Delay

Assume that the end-to-end delay is exponential. This will be true if the delay at one channel dominates the total delay. Since an exponential distribution is completely characterized by its expected value, once we find the expected value of the end-to-end delay, we have determined the distribution.

Consider a communication subnetwork with N nodes and M channels. Denote the channel from node k to node ℓ by the tuple(k,ℓ), where $k,\ell = 1,\ldots N$. Let C denote the set of all channel tuples, i.e. $C = \{(k,\ell) \mid k,\ell = 1,\ldots N\}$.

Let $D_{k\ell}$ = service time at channel (k,ℓ),

T_{ij} = end-to-end delay from node i to node j,

λ_{ij} = Poisson arrival rate of messages at node i destined for node j,

$\gamma_{ij}(k)$ = fraction of i-j traffic passing through node k,

$g_{ij}(k,\ell)$ = fraction of i-j traffic passing through channel (k,ℓ),

$\phi_{ij}(k,\ell)$ = routing variable, fraction of i-j traffic at node k that is routed on channel (k,ℓ),

($0 \leq \phi_{ij}(k,\ell) \leq 1$, $\sum_{\ell} \phi_{ij}(k,\ell) = 1$ for all k)

L_{ij} = average number of i-j messages in the network,

$L_{ij}(k,\ell)$ = average number of i-j messages at channel (k,ℓ).

Little's Formula[LITT61] gives

$$\lambda_{ij}\bar{T}_{ij} = L_{ij} = \sum_{(k,\ell)\in C} L_{ij}(k,\ell) = \sum_{(k,\ell)\in C} \lambda_{ij} g_{ij}(k,\ell)\bar{D}_{k\ell}$$

$$= \sum_{(k,\ell)\in C} \lambda_{ij} f_{ij}(k)\phi_{ij}(k,\ell)\bar{D}_{k\ell}$$

Hence, $$\bar{T}_{ij} = \sum_{(k,\ell)\in C} f_{ij}(k)\phi_{ij}(k,\ell)\bar{D}_{k\ell} \qquad (3.7)$$

where $$f_{ij}(k) = \sum_{\ell=1}^{N} f_{ij}(\ell)\phi_{ij}(\ell,k) \qquad (3.8)$$

If the routing is loop-free, for each origin-destination pair (i,j), Equ.(3.8) can be solved recursively for $f_{ij}(k)$, k = 1,...N. Therefore, the average end-to-end delay for all nodal pairs can be found using Equ.(3.7) and (3.8).

3.4.2 Normal End-to-end Delay

Another approximation is to assume that the end-to-end delay is normal. Since each T_{ij} is the sum of the service times over several channels, by invoking the Central Limit Theorem, the approximation will be good provided the path corresponding to T_{ij} includes many hops.

Note that in section 3.4.1, the derivation does not require that the service time at the different channels be independent. In the following derivation, however, we assume that the service times at different channels

are independent. The notations used are the same as that in section 3.4.1. in addition, let $T_{ij}(t)$ be the pdf of T_{ij}, $T^e_{ij}(s)$ be the Laplace transform of $T_{ij}(t)$, $D_{k\ell}(t)$ be the pdf of $D_{k\ell}$, and $D^e_{k\ell}(s)$ be its Laplace Transform.

Now, $T_{ij} = \begin{cases} D_{ir} + T_{rj} & \text{with probability } \phi_{ij}(i,r) \\ D_{ij} & \text{with probability } \phi_{ij}(i,j) \end{cases}$

Therefore, $T^e_{ij}(s) = \sum_{\substack{r=1 \\ r\neq j}}^{N} \phi_{ij}(i,r) D^e_{ir}(s) T^e_{rj}(s) + \phi_{ij}(i,j) D^e_{ij}(s)$

$$= \sum_{r=1}^{N} \phi_{ij}(i,r) D^e_{ir}(s) T^e_{rj}(s) + \phi_{ij}(i,j) D^e_{ij}(s)[1 - T^e_{jj}(s)]$$

$$T^e_{ij}(s) = \sum_{r=1}^{N} \phi_{ij}(i,r) D^e_{ir}(s) T^e_{rj}(s) \qquad (3.9)$$

since it is assumed that $T_{jj} = 0$.

In particular, if the total message arrival rate at channel(k,ℓ) is $\lambda_{k\ell}$, the capacity is $C_{k\ell}$, and the message length is exponential with mean $1/\mu$, then the total service time at channel(k,ℓ) is exponential with mean $1/\nu_{k\ell}$ where $\nu_{k\ell} = \mu C_{k\ell} - \lambda_{k\ell}$. Therefore, $D_{ir}(s) = \nu_{ir}/(s+\nu_{ir})$ and Equ.(3.9) becomes:

$$T^e_{ij}(s) = \sum_{r=1}^{N} \phi_{ij}(i,r) \frac{\nu_{ir}}{s + \nu_{ir}} T^e_{rj}(s) \qquad (3.10)$$

$E(T_{ij}) = -\frac{d}{ds} T^e_{ij}(s)\Big|_{s=0}$ gives

$$E(T_{ij}) = \sum_{r=1}^{N} \phi_{ij}(i,r) [E(T_{rj}) + 1/\nu_{ir}] \qquad (3.11)$$

$E(T^2_{ij}) = \frac{d^2}{ds^2} T^e_{ij}(s)\Big|_{s=0}$ gives

$$E(T^2_{ij}) = \sum_{r=1}^{N} \phi_{ij}(i,r) [E(T^2_{rj}) + 2E(T_{rj})/\nu_{ir} + 2/\nu^2_{ir}] \qquad (3.12)$$

If the routing variables $\phi_{ij}(k,l)$ correspond to loop-free routing

of the messages (see [GALL77]), then we can find $T^e_{i,}(s)$ easily. From Equ. (3.10), we solve for $T^e_{r,}(s)$ for all r such that $(r,i) \in$, then we solve for $T^e_{k,}(s)$ for all k such that $(k,r) \in$, et ., until we find $T^e_{ii}(s)$. Similarly, we can solve for $E(T_{ii})$ and $E(T^2_{ii})$ easily.

If the routing is not loop-free, then Equ.(3.10) can be re-written in matrix form:

$$^{e}_{(s)} = P^{e}(s)\,^{e}(s) \qquad (3.13)$$

where $^{e}(s)$ is an N by N square matrix with elements $T^e_{i,}(s)$, and $P^e(s)$ is an N by N square matrix with elements $p_{ij}(i,r)\,v_{ir}(s + \,_{ir})$.

Hence $^{e}(s) = [I - P^e(s)]^{-1}$.

3.5 Attempt to Model Message Broadcasts

In a communication network, it is sometimes necessary to broadcast messages from one node to other nodes in the network. After a node receives a broadcast message, copies of this message will be sent to its neighbours. In section 3.2, we have already pointed out some of the inadequacies of Jackson's Queues in modelling a communication network. In this section we shall discuss another inadequacy. We shall show that message broadcasts cannot be modelled by Jackson's Queues.

In Jackson's model, under equilibrium conditions, each individual queue in a network of queues is independent of other queues. We can consider a system of queues , where after being served at one queue, a customer (or message in commuication network context) splits into two and obtains service at two different queues.

In Fig.3.3, nodes 1, 2 and 3 are three queues with exponential service times. Messages arrive at nodes 1 and 2 independently in a Poisson manner. After being served at node 1, each message splits into

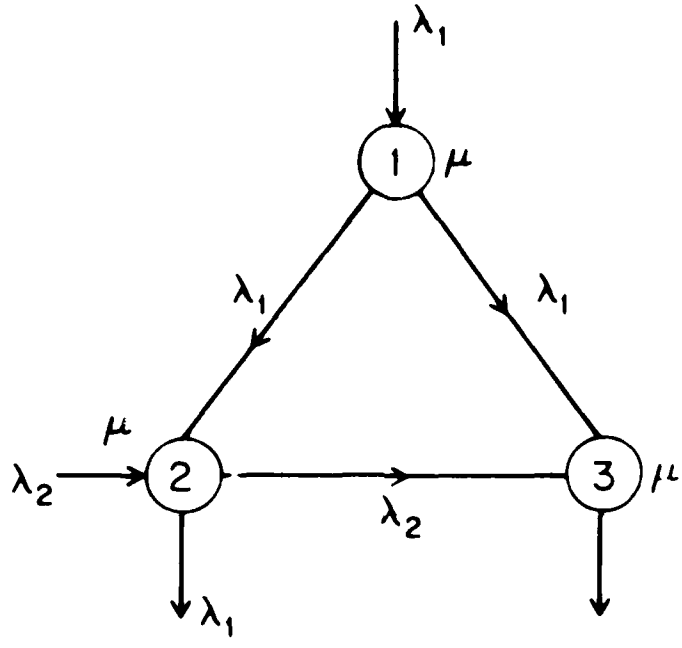


Figure 3.3 Attempt to generalize Jackson's results to Message Broadcasts

By moving the first term on the right to the left hand side and dividing by h, we get a differential equation in $\vec{N}(t)$. Setting $\frac{d}{dt}\vec{N}(t) = 0$ for equilibrium conditions, we get:

$$P[\vec{N} = (m,n)](\lambda_1 + \lambda_2 + 2\mu) = P[\vec{N} = (m-1,n-1)]\lambda_1 + P[\vec{N} = (m-1,n)]\lambda_2 + P[\vec{N} = (m+1,n)]\lambda_1\mu/(\lambda_1+\lambda_2) + P[\vec{N} = (m+1,n-1)]\lambda_2\mu/(\lambda_1+\lambda_2) + P[\vec{N} = (m,n+1)]\mu$$

such that $\sum\sum_{mn} P[\vec{N} = (m,n)] = 1$

In addition, we must satisfy the boundary condition:

$$P[\vec{N} = (0,0)](\lambda_1+\lambda_2) = P[\vec{N} = (1,0)]\lambda_1\mu/(\lambda_1+\lambda_2) + P[\vec{N} = (0,1)]\mu$$

Suppose the compound pmf has a product form, i.e. $P[\vec{N} = (m,n)] = Y^m X^n$, then the equilibrium balance equation gives:

$$YX(\lambda + 2\mu) = \lambda_1 + X\lambda_2 + Y^2X\lambda_1\mu/\lambda + Y^2\lambda_2\mu/\lambda + YX^2\mu \qquad (3.14)$$

while the boundary condition gives:

$$\lambda = Y\lambda_1\mu/\lambda + X\mu \qquad (3.15)$$

Solving Equ.(3.14) and (3.15) simultaneously, we get a quadratic in X:

$$X^2[\lambda\lambda_1\mu^2 + \lambda^2\mu^2] + X[\lambda_1^2\lambda_2\mu - \lambda^2\lambda_1\mu - \lambda^2\lambda_1\mu - 2\lambda^2\lambda_2\mu] + \lambda_1^3\mu + \lambda^3\lambda_2 = 0$$

Solving this quadratic for X, we found that depending on the values of the parameters λ_1, λ_2, μ, we might get imaginary roots for X, and hence imaginary roots for Y. Hence the assumption that $P[\vec{N}(m,n)] = Y^m X^n$ is incorrect, and the compound pmf does not exhibit product form.

Thus Jackson's Queues cannot model message broadcasts. However, using our Independent Queues Model, message broadcasts can be modelled easily. Since we assume that the queueing processes at different channels are independent in equilibrium, and the service time at each channel is

only a function of the total input traffic at that channel. This input traffic may consist of external message arrivals, internal transfers according to some routing strategy and broadcasted messages.

CHAPTER 4

DISTRIBUTED DATABASE MODEL

In this chapter, we are going to describe the DDB model. In section 3.3, we modelled the competition among messages (generated by transactions) for the services of the communication channels. In this chapter, we are also going to model the competition for the services of the local DBMS at each computer site. Thus, in order for a transaction to complete successfully, it not only has to wait for the services of the different communication channels, but also the service of the database management system. This service includes setting locks, reading and writing data items, etc.

Fig. 4.1 shows the basic architecture of a DDB. Database sites are connected to each other via a communication subnetwork. At each database site is a computer running one or both of the software modules: Transaction Module (TM) and Data Module (DM). The TM supervises user interactions with the database while the DM manages the data at each site.

We propose a 5-step approach to model the performance of concurrency control algorithms. This is summarized in Fig.4.2. We shall now describe these five steps in more detail.

4.1 Input Data

Given a DDB managed on an arbitrary communication network, we have to determine the following:

(1) topology of the network, i.e. the connectivity and capacity of links between computer sites.

(2) locations of all copies of files.

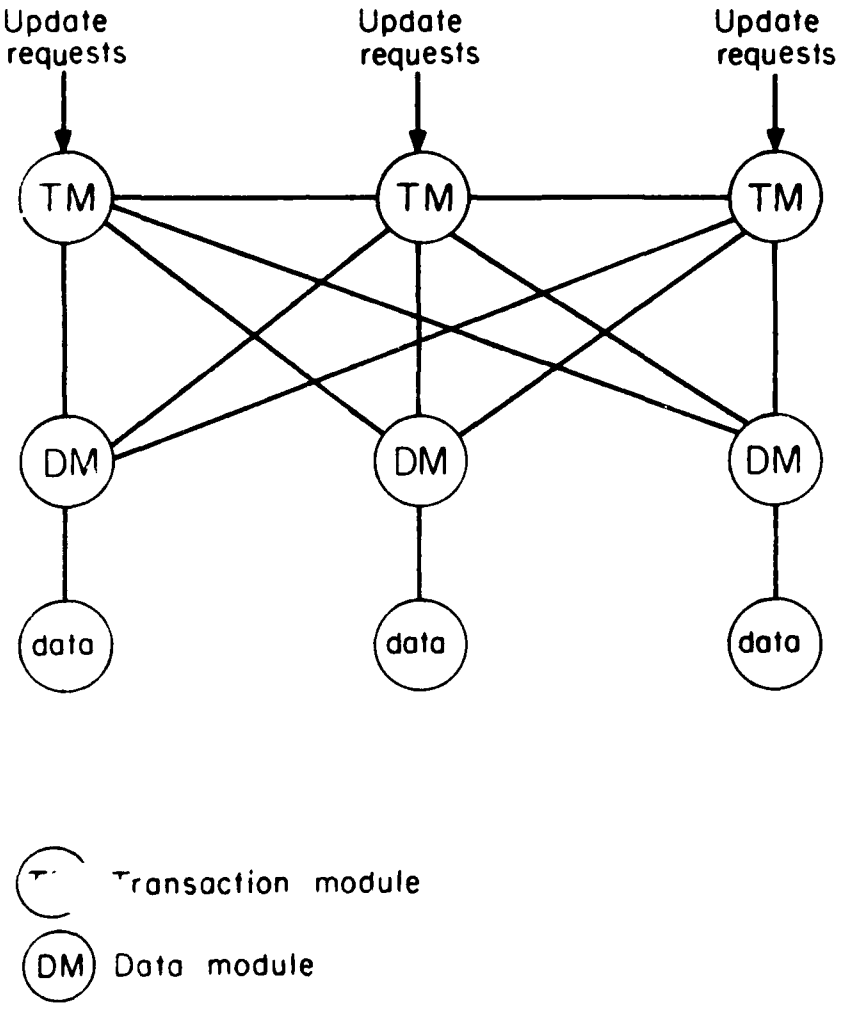


Figure 4.1 Basic Architecture of a DDB

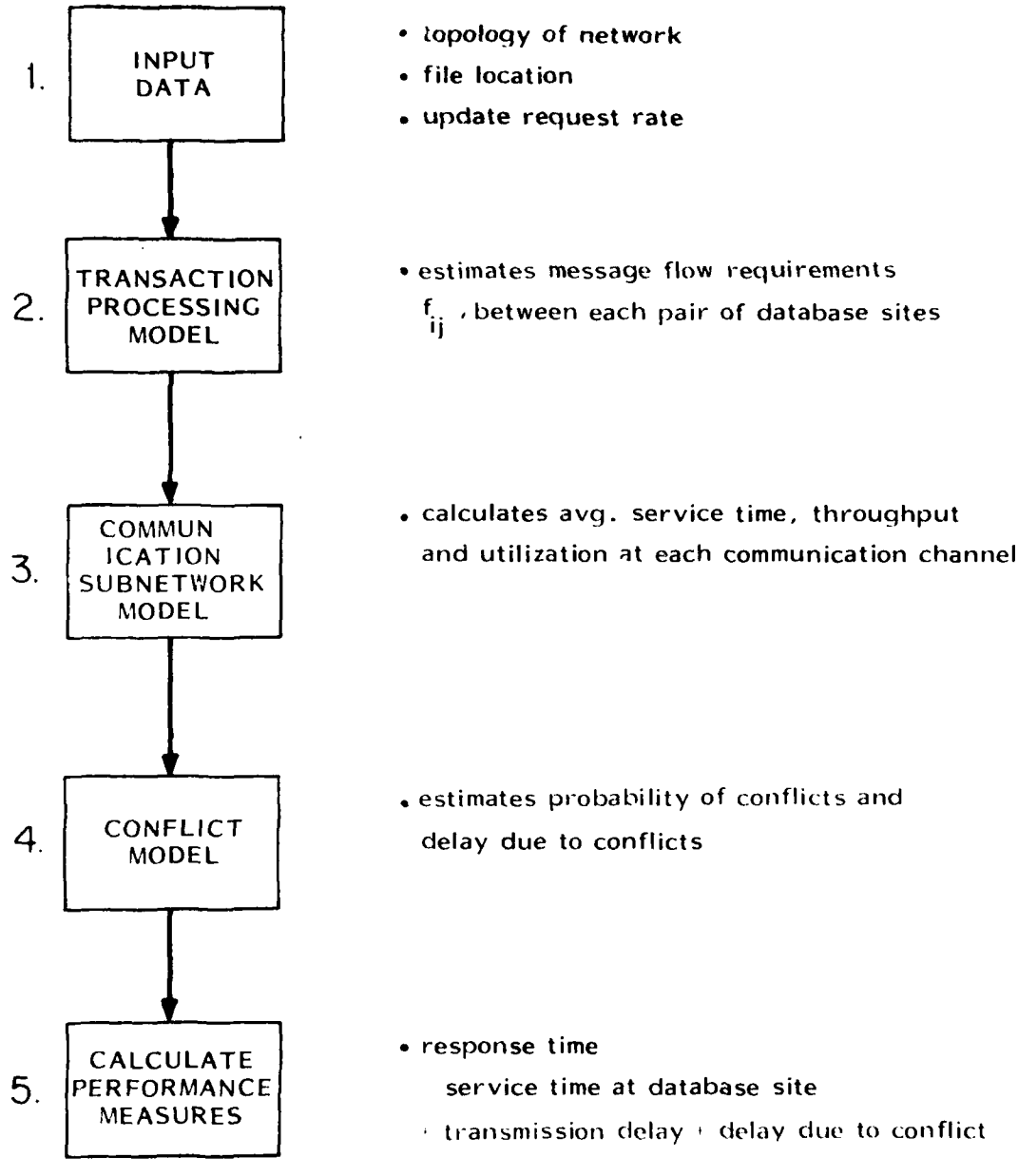


Figure 4.2 Performance Model of a DDB

(3) transaction classes defined by their readsets and writesets.

(4) update request rates of the different transaction classes.

4.2 Transaction Processing Model

Consider a trans. T arriving at database site i and processed by TM_α. Suppose the read set of T consists of data items X,Y and its writeset consists of data items U, V where U = f(X,Y), V = g(X,Y). This update will be performed in two steps, a query processing step and a write step.

(1) Query Processing - TM_α will choose one copy of data item X and Y and devise a query processing strategy that will produce the result at database site i.

(2) Write - The new values of U and V will be written into the database. This is accomplished by the two-phase commit algorithm.

   (i) Pre-commits - TM_α sends new values of U and V to all DM's having copies of U and V, respectively. The DM's then copy the new values to secure storage and acknowledge receipt.

   (ii) Commits - After all DM's have acknowledged, TM_α sends commit messages, requesting the DM's to copy the new values of U and V from secure storage into the database.

Since the network is not perfectly reliable, if TM_α asks the DM's to copy the new values of U and V into the database in one step, it is possible that some copies of U and V have received the updates while other copies have not. This will result in database inconsistency. Two-phase commit is an attempt to prevent such inconsistencies. It is by no means the standard solution. However, it seems to be very popular

among researchers in DDB. (See for example, [BG80] , [GRAY78].)

Step(1) of the Trans. Processing Model, namely Query Processing, is a very hard problem. Since the database is redundant, there are different DM's a transaction can access when it wants to read a certain data item. The TM at the database site where the transaction originates is responsible for choosing the best (in terms of an objective function such as minimization of response time) DM to access. In the case of transactions that access multiple files, the TM must devise a strategy to solve the query.

As is mentioned in section 1.4.2 previous researchers get around the query processing problem by assuming a fully redundant database. in which case all queries will be addressed to the local site and incur zero communication delay. We do not believe this is a realistic assumption, and are confronted with the problem of modelling query processing. In particular, since our major thrust is the performance comparisons of concurrency control algorithms, we need a query processing model that is simple while at the same time not too unrealistic. This is the object of the next chapter.

Using our trans.processing model, we can determine, for each particular transaction, the file transfers, read requests and update messages that are necessary. This information, together with the transaction arrival rates and the file locations, etc., let us generate estimates for f_{ij}, the arrival rate of messages at site i destined for site j.

4.3 Communication Subnetwork Model

Using the message flow requirements between database sites, f_{ij}, and the network topology as input to a routing strategy, such as Gallager's

Minimum Delay Routing Strategy [GALL77], we can determine the total traffic on each channel of the network. We can then find the average service time, utilization and throughput at each channel.

4.4 Conflict Model

This is an important component of the DDB Model and will be discussed in more detail in Chapter 6. Briefly, it allows us to find the probability of conflict between different transactions and the cost of these conflicts.

4.5 Performance Measures

We emphasize the performance measure most visible to the users, namely response time, which is the sum of local processing delay at the database sites, transmission delay and delay due to conflicts.

CHAPTER 5

QUERY PROCESSING

Accessing data distributed at different computer sites necessites transmission over communication links. An advantage of the distributed system is the ability to process and transmit data in parallel at separate points in the network. Since the delay due to communications is substantial, the DBMS must devise an efficient arrangement of local data processing and data transmissions in order to process distributed queries.

5.1 Review of Existing Query Processing Algorithms

There are very few reports of work on distributed query processing. Wong proposed an algorithm that is being implemented in SDD-1 [WONG77]. Hevner and Yao [HY79], proposed a simple algorithm for a special class of queries. These two approaches will be discussed in more detail in the next sections. More recently, Chiu [CHIU79] has devised a dynamic programming solution for certain queries called tree queries. Since the applicability of his method is extremely restricted, we shall not discuss it.

5.1.1 Relational Data Model

In this section, we shall describe the relational data model ([CODD70], [DATE77]), which is the model assumed in both Wong's Algorithm [WONG77] and Hevner and Yao's Algorithm [HY79].

A data model is a class of data structures. All information maintained

in the database is organized as data structures that are instances of this class.

Given n sets D_1, D_2,...D_n, a relation R is a subset of their Cartesian product, i.e. $R \subseteq D_1 \times D_2 \times \ldots \times D_n$, and D_1,...D_n are called the domains of the relation. A database relation is a time-varying subset of the Cartesian product, i.e. $R(t) \subseteq D_1 \times D_2 \times \ldots \times D_n$. The relational model of a DDB assumes that the unit of data distribution is a relation. In each database site, there are one or more relations. A database query performs the operations restriction, projection, join and semi-join in order to retrieve data. (See [CODD70], [WONG77], [CHIU79]). A restriction of R selects rows of R that satisfy specified data conditions, e.g. $D_1 > 100$. A projection of R is formed by deleting some of the domains. For example, $R[D_1,D_2]$ is a projection of R consisting of the first two domains. Consider two relations R(A,B)* and S(C,D). The θ-join of R on A and S on C is $R[A\theta C]S = \{r||s: r \in R \text{ and } s \in S \text{ and } r[A]\theta S[B])\}$, where $r||s = (a,b,c,d)$ if $r = (a,b)$ and $s = (c,d)$; and $\theta = \{<, \leq, >, \geq, =, \neq\}$.

For example:

R

SHIPID	SHIPNAME
1	Kiev
2	JFK
3	QE2
.	.
.	.

S

SHIPID	TONNAGE
4	40
2	50
3	60
1	30
.	.

R[SHIPID = SHIPID]S

1	Kiev	1	30
2	JFK	2	50
3	QE2	3	60

* R(A,B) denotes a relation with only two domains A and B.

The =-join can also be written $R_A \bowtie_C S$. Let R(A,b) and S(b,C) be two relations, then the semi-join $R_B \ltimes_C S = R_B \bowtie_B S[A,B]$, i.e. R⋉S is the join of R and S projected back onto R.

5.1.2 Wong's Algorithm

Wong makes the following assumptions:

(1) Consider subsets of the DDB called "materializations" such that each consists of exactly one copy of each portion of the database. Each materialization is a distributed but non-redundant version of the database. Data retrieval will be performed on one materialization.

(2) Only fragments of a relation, i.e. restrictions, projections, or a combination of them are moved from one site to another.

(3) We are indifferent as to where the final result is produced, so long as it is produced at a single site.

(4) The communication cost as a function of data volume and links is known.

(5) The costs of local operations, i.e. projection, restriction, and join are known.

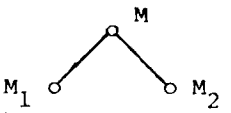
(6) The sizes of resulting fragments after local operations are known.

The algorithm is as follows:

(1) Perform all local operations that are possible. Choose one site to move all fragments to . Denote the set of moves by $M = \{m_1, m_2, \ldots m_n\}$, where each m_k, $k = 1,\ldots m$, is of the form "move fragment x from node i to node j".

(2) Replace M by two sets of moves M_1 and M_2, that are to be executed sequentially with local processing between them. This is represented

graphically as follows:



where it is understood that the left leaf(M_1) precedes the right(M_2) with local processing in between. The criteria for splitting a node N into (N_1,N_2) are:

(i) the combined cost of (N_1,N_2) is less than N alone, and

(ii) the pair (N_1,N_2) is minimum in cost among all pairs satisfying (i).

It is possible that we can continue to split the nodes using the above criteria and in general end up with a binary tree where the leaf nodes represent all the moves that are to be undertaken. The moves within one leaf can be made in parallel,and the leaves are executed sequentially from left to right.

(3) The algorithm stops when no further node splitting is possible.

Note that Wong's Algorithm is a greedy algorithm. It looks for immediate gains and will give us a local optimum but not necessarily a global one.

5.1.3 Hevner and Yao's Algorithm

The following assumptions are made:

(1) & (2) same assumptions as Wong's algorithm.

(3) The result is to be produced at the node where the query originates.

(4) The communication cost between any two nodes is defined as a linear function $C(X) = c_0 + c_1X$ where X is the amount of data transmitted, and c_0 and c_1 are constants. In other words, the topology of the network and the queueing effects are ignored. The cost is measured in units of time.

(5) The costs of local operations, i.e. projections, restrictions, etc. are negligible compared to the communication costs.

(6) It is assumed that after initial local processing, each relation (or file)

accessed in the query contains only one domain - the common joining domain. When file i, of size S_i is processed with file j, the resulting file has size $S_i p_j$* where p_j, the selectivity parameter of file j is between 0 and 1. The selectivity parameter is cumulative, and if file i is processed with both file j and file k, the resulting file will have size $S_i p_j p_k$.

The data transmission containing the transmission of relation i, R_i, to the result node is called the schedule of R_i. Define schedule response time r_i as the time from the start of the transmission until R_i (or a processed version of it) is received at the result node. Define minimal schedule response time $\hat{r}_i = \min r_i$ where the minimization ranges over all possible schedules. The response time of a query retrieval strategy r is defined as $r = \max_{1 \leq i \leq m} r_i$ where m is the number of relations in the query, and total time t is defined as the sum of all transmission costs for the strategy.

Hevner and Yao proposed two algorithms: a parallel strategy to minimize r and a serial strategy to minimize t.

The parallel strategy uses the initial feasible solution of sending each relation directly to the result node as a starting strategy. It then searches for cost beneficial data transmissions by trying to join small relations to large relations. The strategy is described as follows:

(1) Relations R_i are ordered so that $S_1 \leq S_2 \leq \ldots \leq S_m$.

(2) Repeat steps (3) to (4) for i = 1 to m.

(3) Find the minimal schedule response time $\hat{r}_i$. All relations R_j where $j < i$ are checked for potential data transmission to R_i and the transmission that produces the greatest reduction in r_i is integrated into the strategy. For the data transmission from R_j to R_i, the

* While Hevner and Yao[HY79] did not state it, the implicit assumption is that $S_i p_j = S_j p_i$.

[illegible]

(4) All files accessed by the query have the same size.

(5) The selectivity parameter of all files is one. Therefore, when two or more files are processed together, the resulting file has the same size as one of the original files.

Assumptions (3),(4) and (5) imply that the cost of a query processing strategy is a function of the communication links employed in the strategy, irrespective of the volume of traffic on these links.

In addition, we are restricting all file processing to be performed at the result node or at the nodes where the file copies accessed by the query are located. This restriction is necessary in order for the MST Algorithm to be optimal. If file processing can be performed at any node in the communication subnetwork, then in order to find the optimal strategy, we have to solve a Steiner Problem, which is a much harder optimization problem than the MST. This will be discussed in more detail in the next section.

The MST Algorithm finds the optimal query processing strategy that minimizes the total communication costs. Recall the communication subnetwork model described in section 3.3. The average delay T for all messages in the network is given by Equ.(3.5):

$T = \frac{1}{\gamma} \sum_i \frac{\lambda_i}{\mu C_i - \lambda_i}$, where the summation is taken over the set of all channels.

Hence, $\frac{\partial T}{\partial(\lambda_i/\mu)} = \frac{C_i}{\gamma(C_i - \lambda_i/\mu)^2}$ = the incremental delay for all messages in the network per unit increase in flow at channel i, provided the increase in flow is small compared to the existing traffic at the channel. If we let $\frac{\partial T}{\partial(\lambda_i/\mu)}$ be the communication costs on channel i, the MST Algorithm will output a strategy that minimizes $\sum_i \frac{\partial T}{\partial(\lambda_i/\mu)}$. The cost of a strategy is proportional to $\sum_i \frac{\partial T}{\partial(\lambda_i/\mu)}$ because of unit file size. Therefore, the MST strategy

will minimize the incremental delay for all messages in the network due to a particular query. Obviously, other communication costs can be used.

There are two cases to consider: non-redundant files and redundant files.

5.2.1 Non-redundant Files

In this case, each file accessed by the query has only one copy maintained in the database.

Define a directed tree as a directed graph without a circuit, for which the outdegree of every node is unity: the outdegree of the root node being zero. Note that our definition of a directed tree is different from the usual definition (see,e.g. [CHRI75]) in that our directed links point towards the root node, rather than outwards from the root node.

We next describe the MST Algorithms for non-redundant files. There are two algorithms: (1) the MST1 Algorithm restricts all file processing at the node set N, where N is the set of nodes consisting of the result node R and the nodes where the file copies accessed by the query are located, (2) the MST2 Algorithm allows file processing at all nodes.

The MST1 Algorithm

(1) Using the communication costs on the links of the communication sub-network as input to a Shortest Path Algorithm, find the shortest paths between every pair of nodes in N. In general, the shortest path from node i to node j will have different length from the node j to node i path.

(2) Construct a fully connected directed graph G with node set N, and

links weights equal to the shortest path lengths between nodal pairs as calculated in Step (1).

(3) Find the minimum weight directed spanning tree of G using mode R as the root node of the tree.

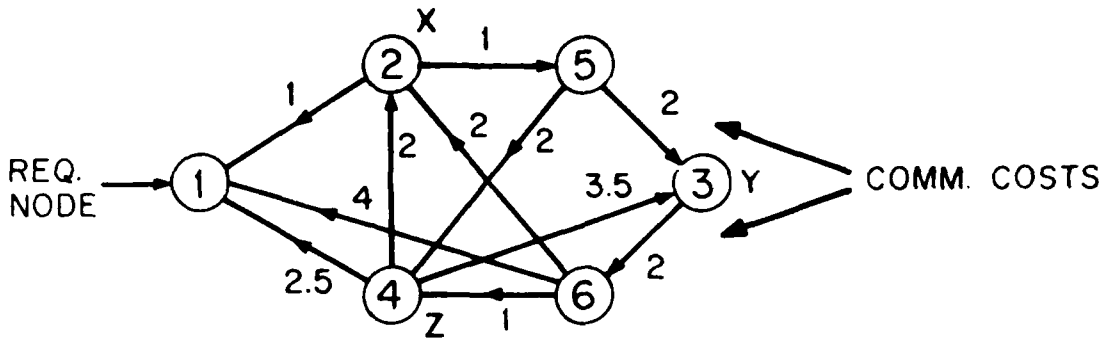
(4) Each file is moved to the result node R using the directed path dictated by the MST. When two paths intersect, the two corresponding files are processed together, resulting in one file.

The algorithm is best illustrated by an example. Consider the six-node communication network shown in Fig.5.1(a). A query originating at node 1 accesses files X,Y and Z at nodes 2,3 and 4 respectively. The MST1 Algorithm says that we shall first find the shortest paths between every pair of nodes in the set of nodes {1,2,3,4}. We then construct a fully connected graph with node set {1,2,3,4} and link weights equal to the shortest path lengths (See Fig.5.1(b)). The MST consists of the directed links (3,4), (4,2) and (2,1) (See Fig.5.1(c)). This means that file Y should be sent to node 4, to be processed with file Z. The resulting file is then sent to node 2, to be processed with file X, and the final result is then sent to node 1.

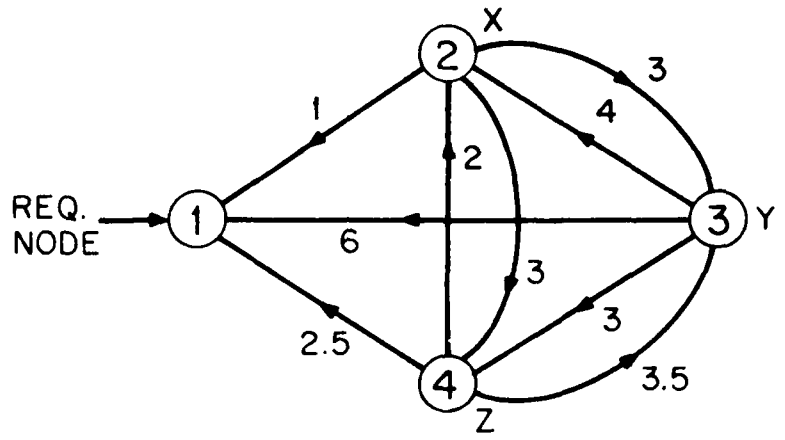
The correctness of the MST1 Algorithm is based on the following theorem.

Theorem 5.1 Under the assumptions of the MST1 Algorithm, each query processing strategy is dominated by (i.e. more costly than) a spanning tree strategy with node R, the result node, as the root node.

Proof: The specification that R is the root node is necessary since we want the result produced at R. Consider a fully connected directed

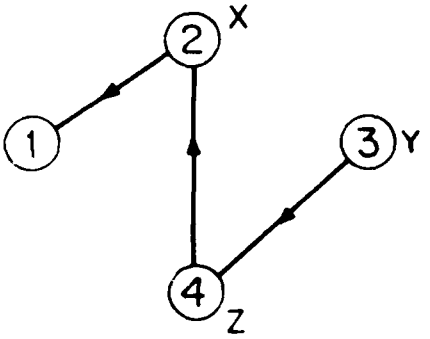


(a) Communication Subnetwork



(b) Graph G with Node Set N

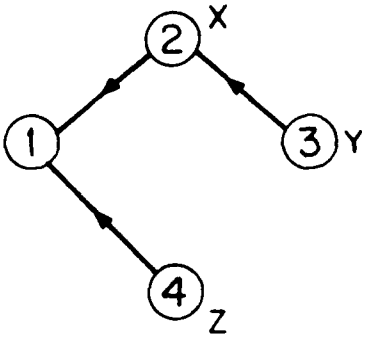
MIN. SPANNING TREE (MST)



Total cost = 6

(c) MST

MIN. DIST. TREE (MDT)



Response time = 5

(d) MDT

Figure 5.1 Examples of the MST and MDT Query Processing Algorithms

graph G with node set N, i.e. the set of nodes consisting of R and the nodes where the files are located. Every query strategy corresponds to a subgraph of G. If the strategy does not correspond to a tree, some node in N will have an outdegree greater than one, say two. This means that the file located at node i will be sent out twice through two different links. This is obviously inferior to just sending the file through one of the two links.

Q.E.D.

The MST1 Algorithm is optimal since the MST is the least costly among all spanning trees.

The MST2 Algorithm

(1) Using the communication costs on the communication links as the weights of the links, find a minimum weight tree that spans the node set N, with node R as the root of the tree.

(2) Each file is moved to the result node R using the directed path dictated by this minimum weight tree. When two paths intersect, the two corresponding files are processed together, resulting in one file.

Step (1) of the MST2 Algorithm corresponds to finding a solution to the Steiner Problem, i.e. finding a minimum weight tree that spans a subset of the nodes of a graph. The Steiner Problem is a much harder problem than the MST Problem. Therefore, in this thesis, we shall employ the MST1 Algorithm, i.e. restricting all file processing to the node set N.

5.2.2 Redundant Files

In this case, each file accessed by the query may have one or more

copies maintained in the database.

By inventing artifical file nodes and artifical links of weight W connecting each file node with its copy locations, the MST Algorithm that we have developed for the non-redundant case can be extended to be used for the redundant case.

Consider Fig.5.2, where we have a request accessing two files X and Y. There are copies of X at nodes 2 and 3, and copies of Y at nodes 4 and 5. Note that we have created directed artifical arcs (X,2),(X,3), (Y,4) and (Y,5) with weights W. The direction of the artifical arcs for file X(orY) ensures that only one of them will be included in the optimal strategy. This is necessary since only one of the copies of X(or Y) will be accessed. The weight W can be chosen to be zero, or may be assigned to be different for different artifical links, to reflect the different costs of accessing a file at different copy locations.

A strategy to satisfy a query originating at node 1 and accessing files X and Y will be represented by a tree spanning node 1 and the artifical nodes X and Y, but not necessarily spanning the whole node set. The optimal strategy corresponds to the minimum weight tree of this type, i.e. a Steiner Tree.

In general, if node i is the requesting node and files X,Y, ...Z are accessed in the query, then the optimal strategy corresponds to finding the minimum weight tree spanning the node set U, where $U = \{i,X,Y,...Z\}$.

The Steiner Problem can be solved by solving a number of MST's. For example, if we want to solve the Steiner Problem for the node set U in a graph with node set V, then we have to solve MST for all subgraphs of

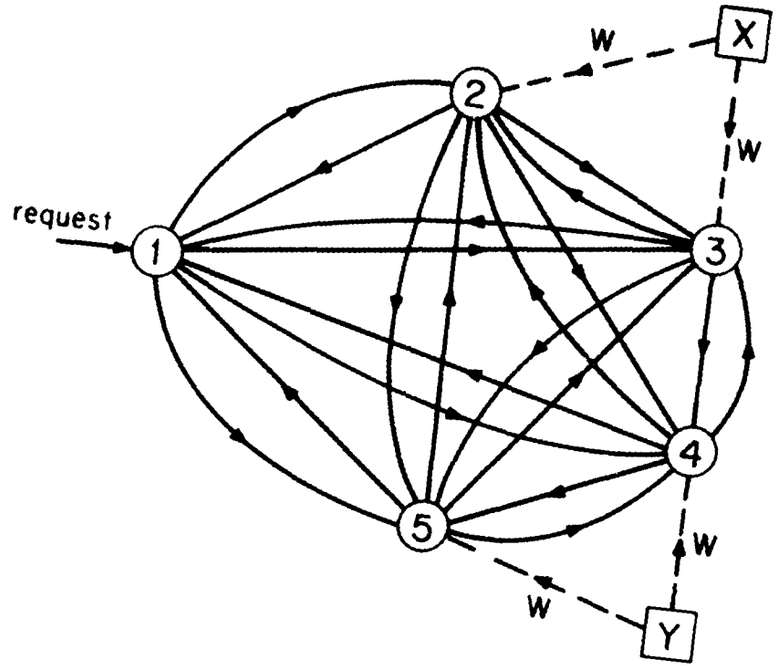


Figure 5.2 Query Processing with Redundant Files
-- The Steiner Algorithm

V that contain U[HAKI71] . Unfortunately, the number of subgraphs is large. However, because of the special structure of the query processing problem, only some of these subgraphs will correspond to query processing strategies. Consider the example shown in Fig.5.2; all we need to consider is the four subgraphs shown in Fig.5.3.

In general, if we have m files and n copies of each file, the number of subgraphs to consider is n^m.

This artifical file node and artificial link technique can be used to generalize Wong's Algorithm [WONG77] to redundant databases. Hevner and Yao's Algorithm [HY79],on the other hand,cannot be generalized easily, since they assumed the same communication costs between each pair of nodes and do not allow us to differentiate the costs of accessing different copies.

5.3 The Minimum Distance Tree Algorithm

The assumptions of the Minimum Distance Tree (MDT) Algorithm are the same as that of the MST Algorithm and are listed in section 5.2. While the MST Algorithm minimizes the total communication costs, the MDT Algorithm minimizes the maximum of the communication costs for sending each file to the requesting node. In particular, if we designate the transmission delays on the communication channels as the communication costs, the MDT Algorithm will find the query processing strategy corresponding to minimum response time.

The MDT Algorithm

(1) Construct a directed graph H of the communication subnetwork. The nodes of H are the nodes of the communication subnetwork and the links

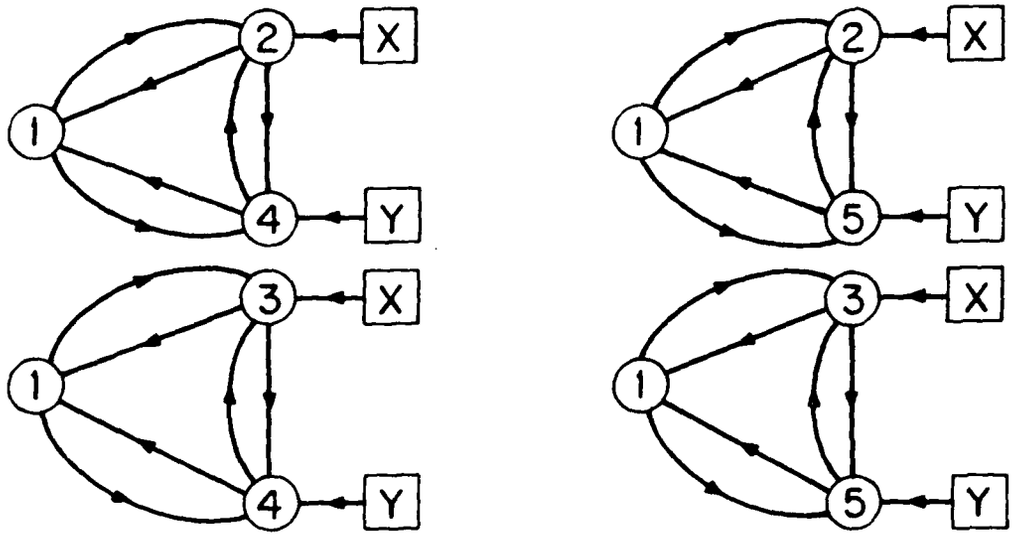


Figure 5.3 The Four Subgraphs to be Considered in the MST Solution of the Steiner Algorithm

of H are the communication channels with weights equal to the communication costs of the channels.

(2) Create artifical file nodes for all files accessed by the query, and create artifical links of zero weight connecting each file node and its copy loctions. These artifical links should be directed outwards from the file nodes.

(3) Find the shortest directed paths from all file nodes to the requesting node. These shortest paths correspond to the paths taken for each individual file to reach the requesting node.

Note that the MDT Algorithm is the same for both redundant and non-redundant databases. In addition, since we are only interested in finding the shortest path for a file to reach the requesting node, the restriction that all file processing must be performed at the node set N can be removed. (Recall that N consists of the result node and the nodes containing copies of the files accessed by the query).

An example of the MDT Algorithm is shown in Fig. 5.1(a), in which a query originating at node 1 accesses files X,Y and Z at nodes 2,3 and 4 respectively. The shortest paths from nodes 2,3 and 4 to node 1 are as shown in Fig. 5.1(d). These are the paths taken by the files to reach the result node. The MDT Algorithm is based on the following theorem.

Theorem 5.2 : Under the assumptions of the MDT Algorithm, the MDT Algorithm will minimize the response time.

Proof: Consider a query accessing files 1,2,...n. The response time, by definition, is $\max_{1 \le i \le n} F_i$, where F_i = time it takes a file i to reach the requesting node. Therefore, to minimize the response time, we have to min.$(\max_i F_i)$. The MDT Algorithm accomplishes this by minimizing each F_i, $1 \le i \le n$.

Q.E.D.

In general, solutions that minimize response times are not unique. So long as the path taken by the file corresponding to the maximum communication delay is unchanged, the paths taken by the other files can be varied without affecting the response time.

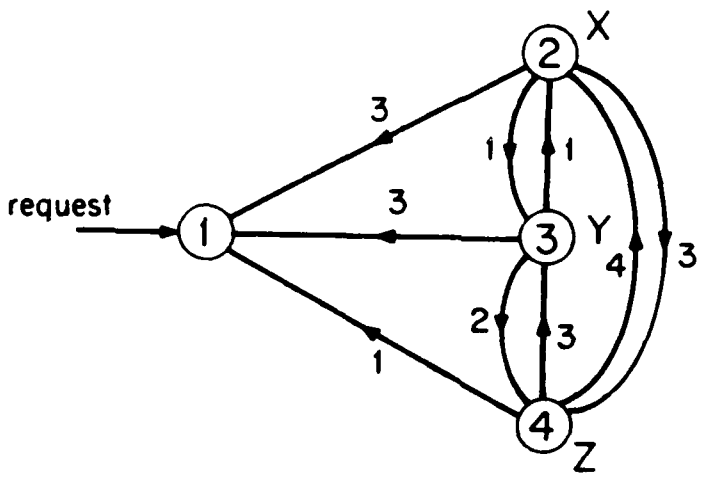
5.4 Comparison of Query Processing Algorithms

Wong's algorithm attempts to solve the most general query processing problem, where (1) files have different sizes, and selectivity parameters are arbitrary and (2) the communication subnetwork is completely general. Unfortunately, this general problem is very difficult and Wong's attempt only resulted in a heuristic giving local optimum.

Hevner and Yao looked at a simpler problem, relaxing requirement(2) by assuming that the communication cost between each pair of nodes depends linearly on the volume of data moved.

The MST Algorithm and MDT Algorithm described in this thesis relax requirement (1) by assuming same size files and selectivity parameters of value one. These algorithms are easy to implement and to analyze.

Note that the MST Algorithm is significantly different from previous work. That it is different from Hevner and Yao's algorithm is obvious, but is it also different from Wong's Algorithm if we relax requirement(1) in implementing Wong's algorithm? The answer is yes. Intuitively, Wong's algorithm does not guarantee a global optimum which the MST Algorithm does, so they cannot be the same. Fig. 5.4 contains a simple counterexample, in which a query tries to access files X,Y and Z. Wong's Algorithm says that the first proposed solution is to send each file directly to the result node (Step 1 in Fig.5.4). We then try to replace this set of moves



Wong's Algorithm

Step 1 : X : 2 → 1
Y : 3 → 1
Z : 4 → 1

Step 2 : Y : 3 → 2
X ⊕ Y : 2 → 1
Z : 4 → 1

Costs 5 units

MST Algorithm

X : 2 → 3
X ⊕ Y : 3 → 4
X ⊕ Y ⊕ Z : 4 → 1

Costs = 4 units

Figure 5.4 Non-equivalence of Wong's Algorithm and the MST Algorithm

by two sets of moves that are less costly than the present set of moves. In this case, we see that it is better to move file Y from node 3 to node 2 first, to be processed with file X, and then send the result to node 1. The move of file Z from node 4 to node 1 remains unchanged (step 2 in Fig.5.4). At this point, no improvement can be made. Wong's Algorithm requires a cost of 5 units, while the MST Algorithm only costs 4 units.

The MST Algorithm and the MDT Algorithm also enjoy the following advantages:

(1) We retain the traditional layering approach, i.e. separating the database from the underlying communication subnetwork. The latter will provide the input for our algorithms: the link costs in the MST and the MDT Algorithms.

(2) The minimum spanning tree problem and the shortest path problem in a directed graph are well understood and there exists efficient algorithms for their solutions.

(3) Under the assumptions we make, the MST and the MDT Algorithm will solve the problem optimally. Other query processing algorithms, for example, Wong's algorithm, only achieve a local optimum. In addition, Wong's algorithm only guarantees that the solution to the query will be available at one site, and is indifferent as to which site it is. The two algorithms proposed in this thesis, on the other hand, guarantee that the result will be at the requesting node.

(4) The MST and MDT algorithms can be easily generalized to accomodate redundant file copies. While Wong's Algorithm can also be generalized using the same technique, Hevner and Yao's Algorithm cannot be generalized easily.

(5) The two algorithms proposed are easy to analyze.

CHAPTER 6

CONFLICT MODELS

In this chapter we shall describe in detail how to model the conflicts between transactions under various concurrency control algorithms. In particular, we shall calculate the probability of conflicts and the delay due to conflicts for Two-phase Locking Algorithms and for Timestamp Ordering Algorithms. The performance of locking algorithms depends very much on the algorithm used to solve the deadlock problem. In section 6.2.1, we analyzed the Ordered Queues Algorithm for deadlock prevention. In section 6.2.2, we analyzed the Prioritized Transactions Algorithm. In section 6.2.3, we calculated the probability of deadlocks when transactions are allowed to wait for each other in an uncontrolled manner, which is the case under Deadlock Detection. Section 6.3 is devoted to SDD-1, a timestamp ordering algorithm.

6.1 Model Assumptions

The following assumptions are made in this chapter:

(1) transaction arrivals are Poison and divided into transaction classes defined by readsets and writesets.

(2) topology of network and location of copies of files are given.

(3) there are two message types, a short type with mean $1/\mu_1$ such as lock requests, requests to read files, etc; and a long message type with mean $1/\mu_2$ such as file transfers, pre-commits, etc. Both types of messages are assumed to have exponentially distributed lengths. Suppose the short and long messages constitute a fraction γ_1 and γ_2 of the total number of messages. We also make the approximation that the

length of all messages is still exponential with mean $\frac{1}{\mu} = \frac{1}{\gamma_1+\gamma_2}\left(\frac{\gamma_1}{\mu_1}+\frac{\gamma_2}{\mu_2}\right)$

(4) Transactions will be processed according to the Transaction Processing Model which consists of two steps: a query processing step and a write step. In the query processing step, the MST1 Algorithm will be used to produce the result of the query at the request node. In the write step, the request node will initiate the two-phase commit algorithm to all nodes containing a copy of the files in the writeset.

(5) Approximate the end-to-end transmission delay of the communication subnetwork as exponential.

(6) One important parameter in locking algorithms is the locking granularity i.e. the size of the unit of the database which is individually locked. Using simulation models, Ries and Stonebraker [RS77] showed that under a wide variety of conditions , coarse granularity gives shorter response times. Therefore, in our performance models, coarse granularity is assumed. In particular, the numerical examples in Chapter 7 assume that whole files are individually locked. This not only simplifies the performance model, but also the data collection necessary to execute the model.

(7) Requests for locks are served in a FCFS manner and the capacity of the queue is infinite.

(8) Two transactions conflict when they try to access the same data item and at least one of them is a write request.

6.2 Two-Phase Locking

When two transactions conflict under the locking algorithm, one of them is made to wait until the other releases its locks. This waiting

incurs delay on the transaction and can be modelled as a queue. Consider, for example, a file X with redundant copies X_1, X_2 and X_3. In the Centralized Locking Algorithm, there is one lock manager for file X and it is located at the central node. All transactions trying to access file X must request a lock on X form this lock manager. In the Distributed Locking Algorithm, there will be three lock managers, one for each redundant copy. Each of these lock managers will be located at the same node as the file copy it manages. A transaction accessing X_i, i = 1,2,3, will request a lock on X_i from the lock manager of X_i. Other files in the database will be managed by their respective lock managers, and each lock manager can be modelled as a queue. The service time at each queue corresponds to the length of time a transaction will hold a lock on the file.

6.2.1 Ordered Queues for Deadlock Prevention

One way to prevent deadlocks is to require transactions to request locks in some universally specified order, i.e. wait for file X first, then Y, then Z, etc. Therefore, the transaction has to obtain service at a network of queues. (See Fig. 6.1). The arrival rate of the different transaction classes to the lock managers will let us calculate the external arrival rates and the routing probabilities for this network of queues. For example, consider two transaction classes with arrival rates λ_1 and λ_3. The first class of transaction accesses both files X and Y, while the second class accesses files X and Z. The total external arrival rate to queue X is $\lambda_1 + \lambda_3$. After obtaining service at queue X, with probability $\lambda_1/(\lambda_1+\lambda_3)$, the request will go to queue Y, and with probability $\lambda_3/(\lambda_1+\lambda_3)$, it will go to queue Z.

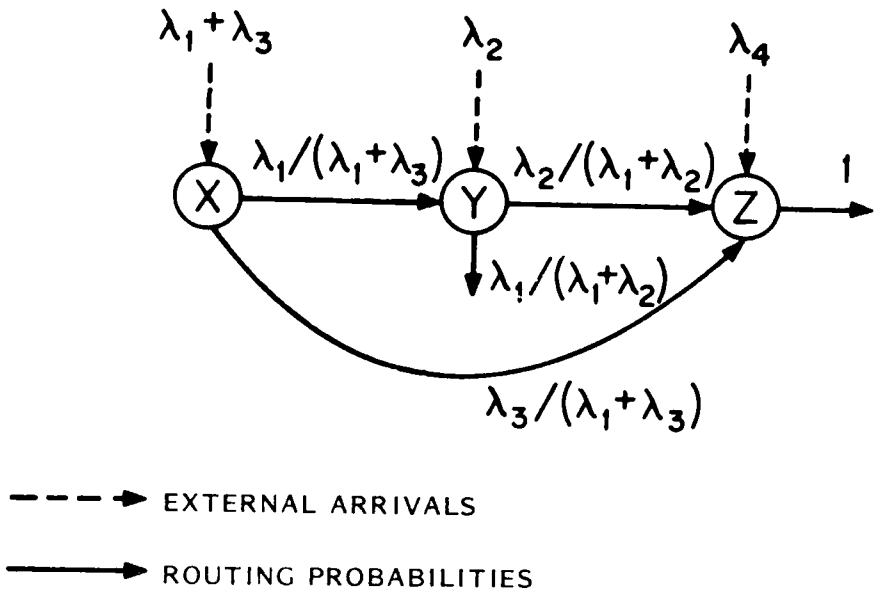


Figure 6.1 Ordered Queues for Deadlock Prevention Modelled as a Queueing Network

The service time at each of the queue in this network of queues, say queue X, corresponds to the time a transaction i accessing file X will hold the lock on X. This service time is the sum of :

(1) transmission delay from lock manager to request node (lock grant),

(2) distributed processing,

(3) transmission delay from request node to lock manager (lock release).

The average service time must be weighted by the arrival rate of the different transaction classes. Thus if the service time due to class 1 transactions T_1 is S_1 and that due to class 2 transaction T_2 is S_2, then the average service time will be $(\lambda_1 S_1 + \lambda_2 S_2)/(\lambda_1 + \lambda_2)$ where λ_1, λ_2 are the arrival rates of T_1 and T_2 respectively.

Our problem is complicated by the fact that a transaction will hold all locks granted until it has finished service. This means that while it is waiting for lock Y, for example, while already holding lock X, all other transactions waiting for lock X will be blocked.

We now make the additional approximation that not only do transactions have to request locks in a specific order, they also have to get served in that order. For example, if transaction i wants to read X and Y, it is required to get a lock on file X, after which it will read X. Then it will get a lock on file Y, and then read Y. This is an approximation because in a real database system, in order to permit more concurrency, as soon as transaction i gets the lock on X, it will often start queueing for the lock on Y.

This approximation is necessary to simplify the model. If we further assume that lock requests arrive in a Poisson manner and service time for each lock request is exponential, the Ordered Queues Algorithm can be approxi-

mated as a Markov Chain. Consider, for example, a two file system shown in Fig. 6.2(a).

There are three possible kinds of lock requests:

(1) lock X then Y: rate λ_1

(2) lock X only : rate λ_2

(3) lock Y only : rate λ_3

Let (m n) denote a state of the system, where n is the number of lock requests both in queue and in service at Y, and m is the number of lock requests at X; $m = b_i$ indicates that lock X is blocked and that there is i requests waiting in queue.

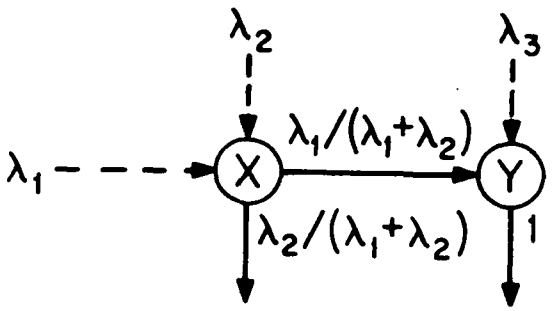
The service rates at X and Y are μ_1 and μ_2 respectively.

The state transition diagram is shown in Fig. 6.2(b). Consider the state (1 0), i.e. one request at queue X, queue Y empty. If a type 1(rate λ_1) or type 2(rate λ_2) request arrives, with rate $\lambda_1+\lambda_2$, we get to the state (2 0). If a type 3 request arrives, we get to state (1 1). If the request at queue X is a type 1 request, after it get served at queue X, it will go to queue Y, while still holding the lock on X, i.e. we get to the state (b_0 1). Thus, even for two files, the model becomes very complex.

An alternative approach is to approximate the network of queues by a series of M/G/1 queues, each corresponding to the transactions waiting for the lock on one particular file.

Consider, for example, a three-file system with lock request rates [λ_1, (X,Y)], [λ_2, (Y,Z)], [λ_3, (X,Z)] and [λ_4, (Z)]. (See. Fig.6.1).

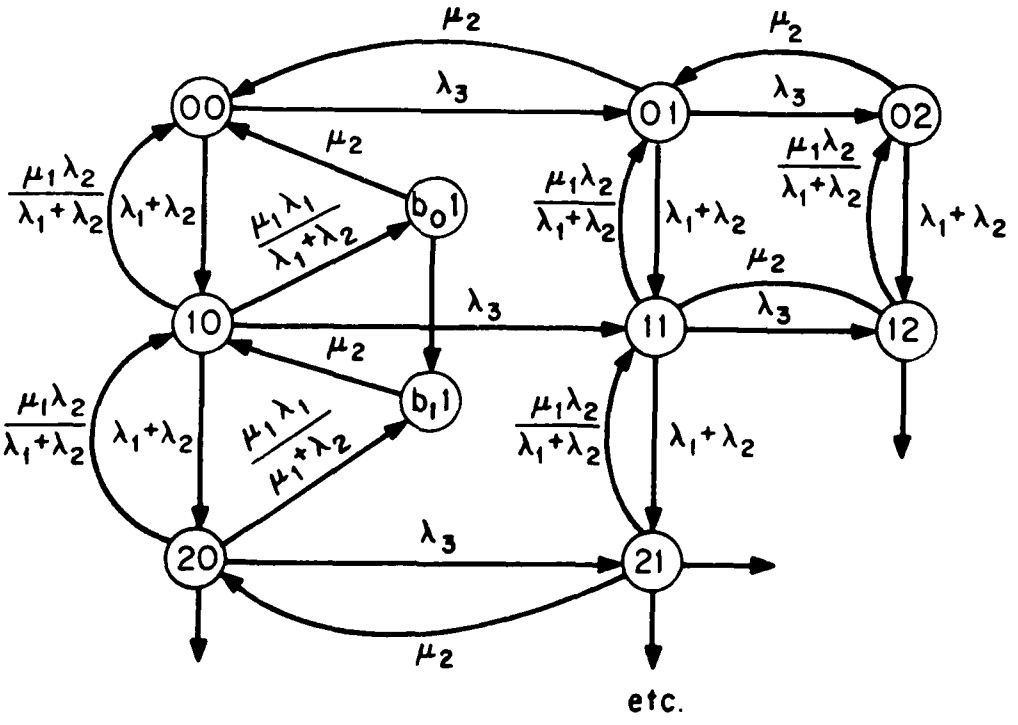
Let b_X, b_Y, b_Z be the in-service time for locking files X, Y, Z individually, i.e. not including delay due to blocking. Let a_X, a_Y, a_Z be the in-service time for locking files X, Y, Z when the files must be locked in the order X → Y → Z, i.e. including delay due to blocking. Let W_X, W_Y,



EXTERNAL ARRIVALS

ROUTING PROBABILITIES

(a) Two file queueing network



(b) Markov model

Figure 6.2 Markov Chain Model of Ordered Queues Algorithm for Two Files

W_Z denote the queueing time and S_X, S_Y, S_Z denote the total service time. i.e. queueing plus in-service time, corresponding to the in-service time of a_X, a_Y, a_Z respectively.

$a_Z = b_Z$ since requests accessing file Z will not be blocked.

$$a_Y = \begin{cases} b_Y & \text{with probability } \lambda_1/(\lambda_1+\lambda_2) \\ b_Y + S_Z & \text{with probability } \lambda_2/(\lambda_1+\lambda_2) \end{cases}$$

since a fraction $\lambda_2/(\lambda_1+\lambda_2)$ of the lock requests at queue Y will release its locks on Y only after they get served at queue Z.

Similarly, $a_X = \begin{cases} b_X + S_Y & \text{with prob. } \lambda_1/(\lambda_1+\lambda_3) \\ b_X + S_Z & \text{with prob. } \lambda_3/(\lambda_1+\lambda_3) \end{cases}$

We can find $f^T_{S_X}(s)^*$, $f^T_{S_Y}(s)$, $f^T_{S_Z}(s)$ by using the Pollaczek-Khinchin transform equation[KLEI75],i.e.

$$f^T_{S_Z}(s) = f^T_{a_Z}(s) \frac{s(1-\rho_Z)}{s - \lambda_Z + \lambda_Z f^T_{a_Z}(s)} \tag{6.1}$$

$$f^T_{S_Y}(s) = f^T_{a_Y}(s) \frac{s(1-\rho_Y)}{s - \lambda_Y + \lambda_Y f^T_{a_Y}(s)},$$

where $f^T_{a_Y}(s) = f^T_{b_Y}(s)\ \lambda_1/(\lambda_1+\lambda_2) + f^T_{b_Y}(s)\ f^T_{S_Z}(s)\ \lambda_2/(\lambda_1+\lambda_2)$

$$f^T_{S_X}(s) = f^T_{a_X}(s) \frac{s(1-\rho_X)}{s - \lambda_X + \lambda_X f^T_{a_X}(s)},$$

where $f^T_{a_X}(s) = f^T_{b_X}(s)\ f^T_{S_Y}(s)\ \lambda_1/(\lambda_1+\lambda_3) + f^T_{b_X}(s)\ f^T_{S_Z}(s)\ \lambda_3/(\lambda_1+\lambda_3)$,

and $\lambda_X = \lambda_1 + \lambda_3$, $\lambda_Y = \lambda_1 + \lambda_2$, $\lambda_Z = \lambda_2 + \lambda_3 + \lambda_4$, $\rho_X = \lambda_X \bar{a}_X$, $\rho_Y = \lambda_Y \bar{a}_Y$, $\rho_Z = \lambda_Z \bar{a}_Z$.

After finding the s-transforms, which must be solved in the order $f^T_{S_Z}(s)$, $f^T_{S_Y}(s)$, $f^T_{S_X}(s)$, we can then find the expected total service time of any transaction. For example, the transaction that acesses files X and Y has an average service time of $\bar{S}_X + \bar{S}_Y$.

Note that the strategy adopted to handle the lock requests from different transactions, say (X,Z) and (Z) transactions, are important. For example, we may give priority to (X,Z) transactions at queue Z, since the (X,Z) transactions hold up more resources. The model we have developed assumes that all requests at queue Z are handled in a FCFS manner. Mathematical models for other queueing disciplines can be developed, although they will probably be more complex.

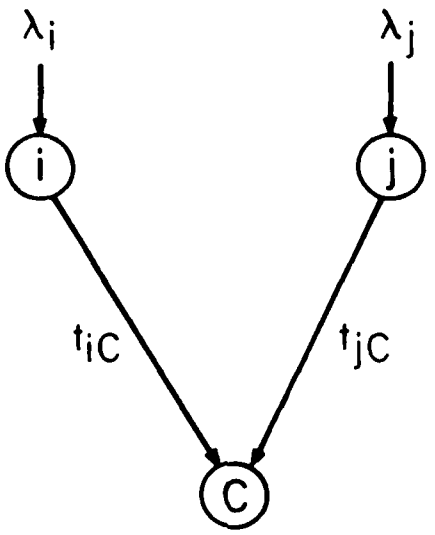
6.2.2 Prioritized Transactions for Deadlock Prevention

We shall now analyze the wait-die and the wound-wait system for deadlock prevention described in 2.4.2.

Suppose two conflicting transaction classes i and j arrive at sites i and j with Poisson rate λ_i and λ_j respectively, and try to obtain a lock on the same data item X at site c. (See Fig.6.3). We would like to find the probability of transaction restarts and the delays associated with them.

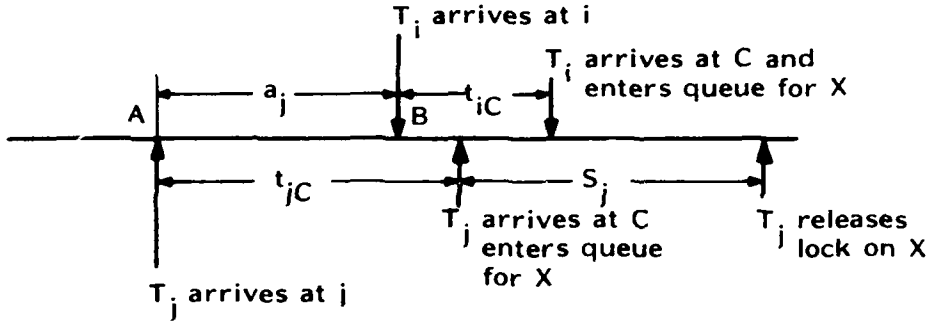
Wait-die

Consider a transaction T_i in transaction class i. T_i will be restarted if it tries to wait for a conflicting transaction which has higher priority. In other words, T_i will be restarted if a conflicting transaction T_j with smaller timestamp (and hence higher priority) than that of T_i arrives at site c before T_i does, and is still in service when T_i arrives. This scenario is depicted in Fig. 6.4(a). T_{ic} is the transmission delay between

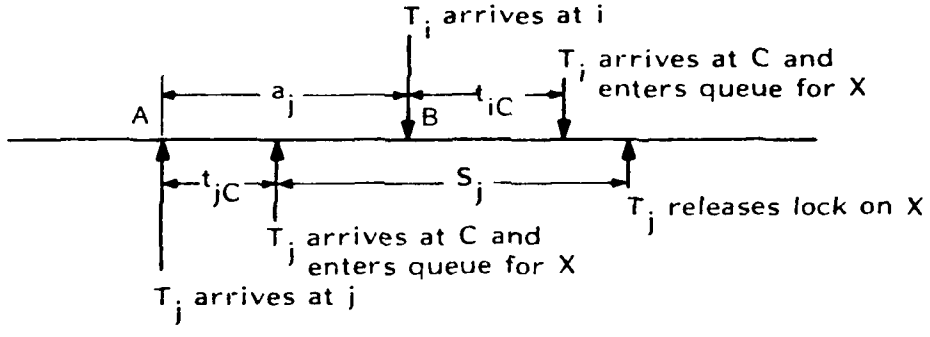


t_{iC} = transmission delay from node i to node C

Figure 6.3 Conflicting Transactions Acessing the Same Data



(a) Case 1: $a_j < t_{jC}$



(b) Case 2: $a_j > t_{jC}$

Figure 6.4 Probability of Restarts under Wait-die

site i and site c, and is exponentially distributed, S_j is the service time of T_j at site c. This includes the queueing time for data item X plus the time T_j holds the lock on X. Point A represents the time at which the most recent transaction T_j arrives at node j. AB therefore represents the length of time one has to go backwards in time until one sees T_j. Since T_j arrives in a Poisson manner, AB is exponentially distributed with mean $1/\lambda_j$.

Hence, P_R = P(T_i is restarted by T_j at node c)

$$= P(t_{jc} < a_j + t_{ic} < t_{jc} + S_j) \qquad (6.2)$$

Equ.(6.2) can be evaluated given a distribution for S_j. To simplify the mathematics, we further assume that S_j is exponentially distributed with mean $1/s_j$.

There are two cases to consider: (1) $a_j < t_{jc}$ (See Fig.6.4(a)), (2) $a_j > t_{jc}$ (See Fig.6.4(b)).

Now $P(t_{jc} < a_j + t_{ic} < t_{jc} + S_j \mid a_j < t_{jc})$

$= P(t_{jc} - a_j < t_{ic} < t_{jc} - a_j + S_j \mid a_j < t_{jc})$

$= P(t_{jc} < t_{ic} < t_{jc} + S_j)$ since $(t_{jc} - a_j \mid a_j < t_{jc}) \sim t_{jc}$*

$= P(t_{ic} > t_{jc})P(S_j > t_{ic} - t_{jc} \mid t_{ic} > t_{jc})$

$= P(t_{ic} > t_{jc})P(S_j > t_{ic})$ since $(t_{ic} - t_{jc} \mid t_{ic} > t_{jc}) \sim t_{ic}$

$$= \frac{\mu_{jc}}{\mu_{ic}+\mu_{jc}} \cdot \frac{\mu_{ic}}{\mu_{ic}+s_j}$$

and, $P(t_{jc} < a_j + t_{ic} < t_{jc} + S_j \mid a_j > t_{jc})$

$= P(0 < a_j - t_{jc} + t_{ic} < S_j \mid a_j > t_{jc})$

$= P(0 < a_j + t_{ic} < S_j)$ since $(a_j - t_{jc} \mid a_j > t_{jc}) \sim a_j$

$$= P(S_j > a_j)P(S_j > t_{ic}) = \frac{\lambda_j}{\lambda_j + s_j} \frac{\mu_{ic}}{\mu_{ic} + s_j}$$

---

*x ~ y means random variables x and y have the same distribution.

$$\text{Hence, } P(T_i \text{ is restarted}) = \frac{\lambda_j}{\lambda_j+\mu_{jc}} \frac{\mu_{jc}}{\mu_{ic}+\mu_{jc}} \frac{\mu_{ic}}{\mu_{ic}+s_j} + \frac{\mu_{jc}}{\lambda_j+\mu_{jc}} \frac{\lambda_j}{\lambda_j+s_j} \frac{\mu_{ic}}{\mu_{ic}+s_j}$$

$$= \frac{\lambda_j}{\lambda_j+\mu_{jc}} \frac{\mu_{ic}\mu_{jc}}{\mu_{ic}+s_j} \left(\frac{1}{\mu_{ic}+\mu_{jc}} + \frac{1}{\lambda_j+s_j}\right) \qquad (6.3)$$

Note that we have simplified the problem, for even if T_i is not restarted by the most recent conflicting transaction T_j, it may be possible that it is restarted by T_j', the transaction that arrives at node j before T_j (see Fig.6.5) or other previous transactions arriving at node j. The probability of these rejections can be calculated in a similar manner. However, since these probabilities are usually very small, we make the assumption that $P(T_i \text{ restarted by } T_j' \text{ or previous conflicting transactions} \mid T_i \text{ not restarted by } T_j) = 0$.

When a transaction T_i is restarted, a lock reject message is sent to the request node which must then terminate the current transaction (in a distributed locking algorithm, this means sending abort messages to all nodes where it has requested locks) and resubmit a new lock request. Since the resubmitted request will retain the original timestamp, it is possible that the new request will conflict with the transaction that kills it before and be killed again (See Fig. 6.6(a)).

Hence, P_{RA} = P(rejection again)

= P(round trip delay to request node plus abort time < remaining service time of T_j)

= $P(t_{ci} + t_{ic} + AT < S_j)$ since S_j is exponential.

AT, the abort time, corresponds to the delay associated with sending abort messages to all nodes where T_i has requested locks and waiting for their acknowledgements. In particular, if AT is exponential with mean $1/\mu_a$, then

$$P_{RA} = \frac{\mu_{ci}}{\mu_{ci}+s_j} \frac{\mu_{ic}}{\mu_{ic}+s_j} \frac{\mu_a}{\mu_a+s_j} \qquad (6.4)$$

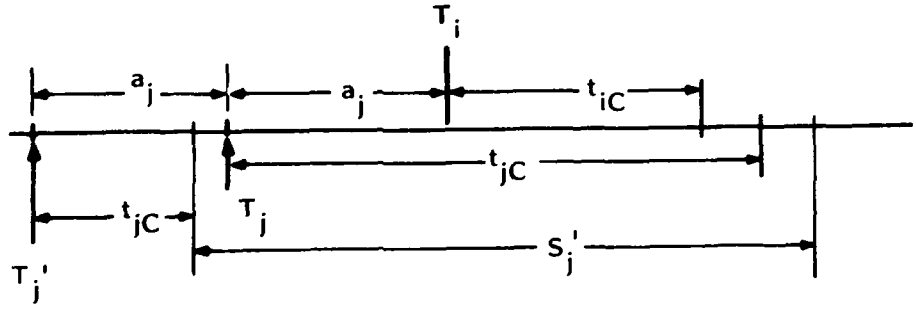
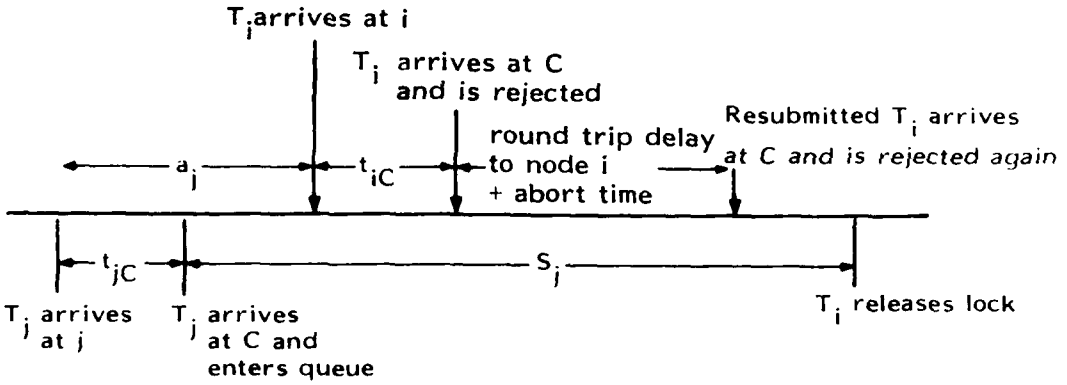
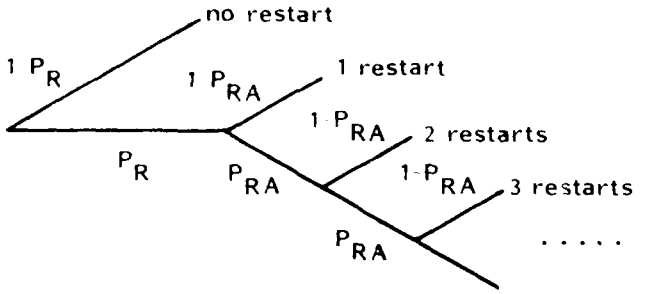


Figure 6.5 Scenario where T_i not restarted by T_j but restarted by T_j'



(a) Probability resubmitted request will be rejected again



(b) Expected number of rejections

Figure 6.6 Finding the Expected Number of Rejections

The probability of this happening yet again is the same because of the self-renewing property of the exponential distribution.

For each transaction T_i,

$$E(\text{number of rejections}) = (1 - P_R) \cdot 0 + P_R(1 - P_{RA}) \sum_{i=1}^{\infty} i P_{RA}^{i-1} \quad \text{(See Fig.6.6(b))}.$$

$$= P_R(1 - P_{RA}) \cdot \frac{1}{(1 - P_{RA})^2}$$

$$= P_R/(1 - P_{RA}) \qquad (6.5)$$

The additional delay due to each rejection = $t_{ci} + t_{ic} + AT$.

Wound-wait

Consider a transaction T_j, it will be wounded by a conflicting transaction T_i if T_i has higher priority and tries to wait for T_j. Fig. 6.7 shows what happens.

Therefore, $P(T_j \text{ is wounded}) = P_W$

$$= P(T_j \text{ is wounded by } T_i, \text{ the most recent conflicting transaction having timestamp earlier than } T_j)$$

$$= P(a_j + t_{jc} < t_{ic} < a_i + t_{jc} + S_j)$$

$$= P(a_i + t_{jc} < t_{ic}) \cdot P(t_{ic} < a_i + t_{jc} + S_j \mid a_i + t_{jc} < t_{ic})$$

$$= P(a_i + t_{jc} < t_{ic}) \cdot P(t_{ic} - a_i - t_{jc} < S_j \mid a_i + t_{jc} < t_{ic})$$

$$= \frac{\lambda_i}{\lambda_i + \mu_{ic}} \cdot \frac{\mu_{jc}}{\mu_{jc} + \mu_{ic}} \cdot \frac{\mu_{ic}}{\mu_{ic} + s_j} \qquad (6.6)$$

Note that we are again ignoring the possibility that T_j may be wounded by previous transactions, i.e. those having earlier timestamps than T_i. Formally, we are assuming that $P(T_j \text{ wounded by previous transactions} \mid T_j \text{ not wounded by } T_i)$ = 0.

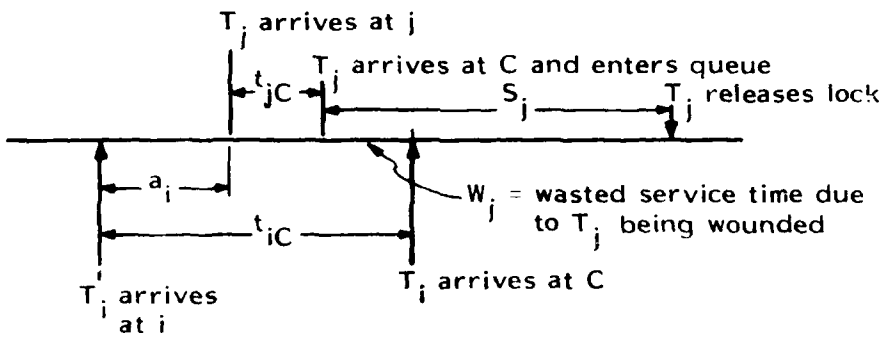
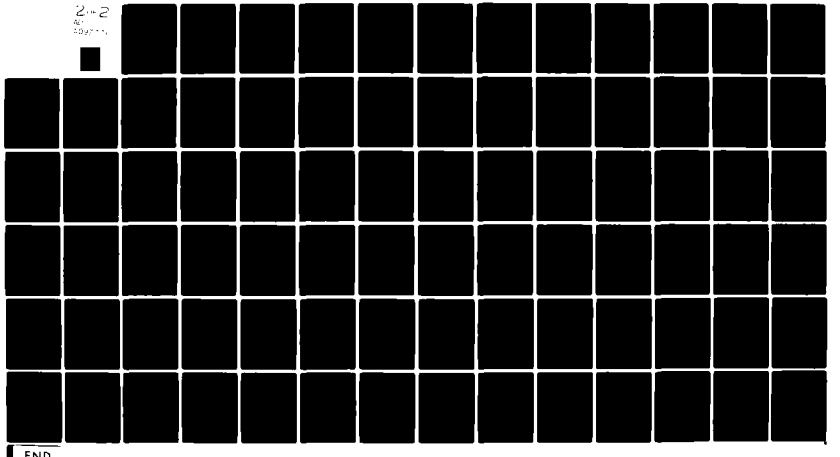


Figure 6.7 Probability T_j is wounded by T_i under Wound-wait

AD-A097 771 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR INFORMA--ETC F/G 9/2
PERFORMANCE MODELS OF DISTRIBUTED DATABASE SYSTEMS.(U)
JAN 81 V O LI N00014-77-C-0532
UNCLASSIFIED LIDS-TH-1066 NL



END
DATE FILMED
5-81
DTIC

Now, we make the additional assumption that all wounded transactions are eventually restarted, i.e. when a wound message arrives, the transaction has not reached the commit phase of the two-phase commit. In this case, the wounded transaction will be resubmitted, with the original timestamp. In wound-wait, in contrast to wait-die, even if the resubmitted transaction conflicts with the transaction that wounds it, it will not be wounded again. This is because the resubmitted transaction is now the requestor and, having lower priority, it is allowed to wait.

For each transaction T_j, E(number of restarts) = P_W (See Equ.(6.6)).

The additional delay due to each wound (See Fig.6.7)

$$= t_{cj} + t_{jc} + W_j$$

$$= t_{cj} + t_{jc} + (t_{ic} - a_i - t_{jc} \mid a_i + t_{jc} < t_{ic} < a_i + t_{jc} + S_j)$$

$$= t_{cj} + t_{jc} + \min(t_{ic}, S_j) \qquad (6.7)$$

Given $t_{ic} - a_i - t_{jc} > 0$, $t_{ic} - a_i - t_{jc}$ has the same distribution as t_{ic}. W_j is thus an exponential restricted to be less than S_j, another exponential. The derivation in Appendix I shows that W_j has the same distribution as $\min(t_{ic}, S_j)$.

Note that in using prioritized transaction for deadlock detection, a transaction can start queueing for all files that it wants to access simultaneously, in contrast to the case of ordered queues where it has to wait for file X first, then Y, then Z, etc. Suppose a transaction has to lock both files X and Y, then the time it has to wait until both locks are granted, provided the lock requests are not rejected, is given by $D = \max.(w_X, w_Y)$ where w_X, w_Y are the queueing time at queues X and Y respectively.

In particular, if the service time (not including queueing) at the queues

are exponential with means $1/\mu_X$ and $1/\mu_Y$ respectively, then E(D) is found to be $\frac{\rho_X}{\mu_X(1-\rho_X)} + \frac{\rho_Y}{\mu_Y(1-\rho_Y)} - \frac{\rho_X\rho_Y}{\mu_X(1-\rho_X) + \mu_Y(1-\rho_Y)}$ in Appendix II, where $\rho_X = \lambda_X/\mu_X$, and $\rho_Y = \lambda_Y/\mu_Y$.

Similarly,if a transaction has to request locks on files W, X,...Z, then D = max.$(w_W, w_X, \ldots w_Z)$ and again E(D) is given by an expression derived in Appendix II.

6.2.3 Probability of Deadlocks

Another way to solve the deadlock problem is deadlock detection. As is mentioned previously (section 2.4.1), this is practical only for Centralized Locking Algorithms. Periodically, the deadlock detector, which is located at the central node, will construct the waits-for graph and determine if there are any deadlocks. When a deadlock is detected, one of the transactions is restarted to break the deadlock.

Therefore, one important parameter in our conflict model is the probability of deadlocks. For each transaction, we must find (1) the probability that it will be involved in a deadlock with other transactions, and (2) the expected delay due to this deadlock.

In the following analysis, we shall consider deadlocks involving only two transactions. In addition, we make the following assumptions:

(1) The Transaction Processing Model says that transactions will be processed in two steps: a query processing step, and a write step. Thus, when a transaction T arrives at the central node, it will request locks on all files in its readset. It is assumed that T starts queueing at all files in its readset simultaneously. T next performs query processing,

after which it will request locks on all files in its writeset. Again T will start queueing at all files in its writeset simultaneously.

(2) The elapsed time between when T requests locks on its readset and writeset is assumed to be exponentially distributed.

Consider Fig. 6.8(a) in which two classes of transactions try to obtain locks from the central node. Class 1 transactions have a readset consisting of file X and a writeset consisting of file Y while class 2 transactions have file Y as the readset and file X as the writeset, thereby creating a potential deadlock. There are two cases to consider: (1) Class 1 transaction arrives at the central node first, and (2) Class 2 transaction arrives first.

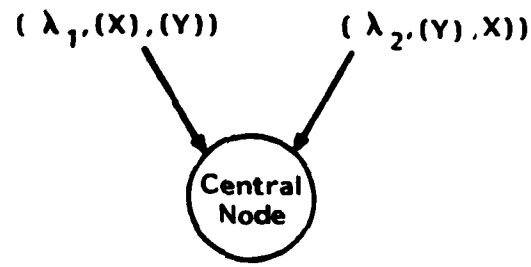
Suppose a Class 1 transaction T_1 arrives first (Fig. 6.8(b)), with probability $\lambda_1/(\lambda_1+\lambda_2)$. According to the Transaction Processing Model, when T_1 arrives at the central node, it will request to lock its readset, namely file X.Upon obtaining the lock on X, it will perform query processing, which in this case corresponds to reading file X. Then it will request a lock on file Y, its writeset. The time between when T_1 requests locks on X and Y is represented by AB in Fig.6.8(b).

Any Class 2 transaction T_2 arriving after time A will try to access file Y and must wait until T_1 is completed. In addition, if T_2 arrives at time C before T_1 requests the lock on file Y, i.e. during the period represented by AB in Fig.6.8(b), then when T_1 wants to access file Y, it must wait for T_2. A deadlock is created and the probability of deadlock, P_{DL}, is given by:

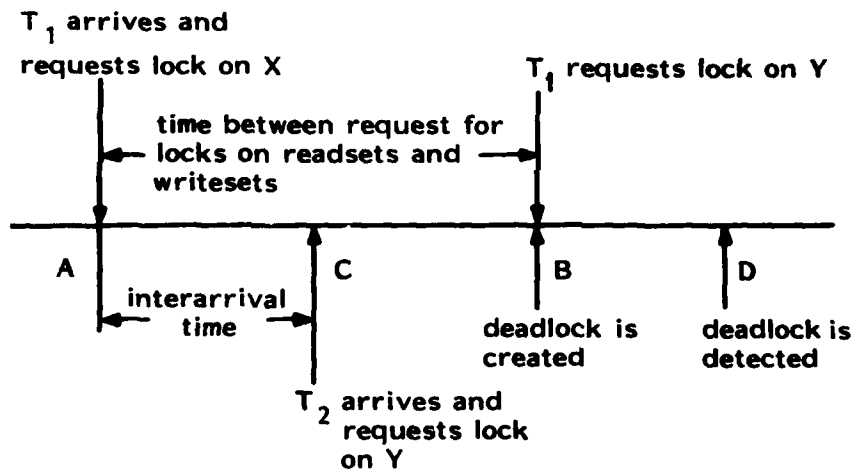
$$P_{DL} = P(AC < AB) = \lambda_2/(\lambda_2+\mu_1) \qquad (6.8)$$

where $1/\mu_1 = E(AB)$ and AB is exponentially distributed.

The deadlock detector at the central node constructs the waits-for graph periodically and takes time BD (Fig. 6.8(b)) to detect the deadlock.



(a) Two conflicting classes of transactions arrive at central node



(b) A deadlock is created

Figure 6.8 Probability of Deadlocks

If the waits-for graph is constructed every S seconds, say, then E(BD) = S/2 seconds. After the deadlock is detected, the deadlock detector will break the deadlock by restarting one of the deadlocked transactions. In this case, in order to minimize wasted resources, it will restart T_2 since T_2 has barely started while T_1 has already finished its query processing step. Therefore, given T_1 arrives first, expected delay for T_1 due to deadlock = E(BD) = S/2, while expected delay for T_2 due to deadlock = E(CB) + E(BD) = E(AB - AC | AB > AC, + E(BD) = E(AB) + E(BD) = $1/\mu_1$ + S/2.

The symmetric situation of T_2 arriving at the central node first (with probability $\lambda_2/(\lambda_1+\lambda_2)$ is completely analogous. Thus, given T_2 arrives first, P(deadlock) = $\lambda_1/(\lambda_1+\mu_2)$ where $1/\mu_2$ is the expected elaped time between when T_2 requests locks on its readset and its writeset. For T_1, E(delay due to deadlock) = S/2.

Hence, P(deadlock between T_1 and T_2)

$$= P(\text{deadlock} \mid T_1 \text{ arrives first})\, P(T_1 \text{ arrives first}) + P(\text{deadlock} \mid T_2 \text{ arrives first})\, P(T_2 \text{ arrives first})$$

$$= \frac{\lambda_2}{\lambda_2+\mu_1}\frac{\lambda_1}{\lambda_1+\lambda_2} + \frac{\lambda_1}{\lambda_1+\mu_2}\frac{\lambda_2}{\lambda_1+\lambda_2} \qquad (6.9)$$

and E(delay due to deadlock for T_1)

$$= E(\text{delay for } T_1 \mid T_1 \text{ arrives first, deadlock occurs}) \cdot P(T_1 \text{ arrives first, deadlock occurs}) + E(\text{delay for } T_1 \mid T_2 \text{ arrives first, deadlock occurs}) \cdot P(T_2 \text{ arrives first, deadlock occurs})$$

$$= \frac{S}{2}\frac{\lambda_1}{\lambda_1+\lambda_2}\frac{\lambda_2}{\lambda_2+\mu_1} + \left(\frac{1}{\mu_2}+\frac{S}{2}\right)\frac{\lambda_2}{\lambda_1+\lambda_2}\frac{\lambda_1}{\lambda_1+\mu_2} \qquad (6.10)$$

Similarly, E(delay due to deadlock for T_2)

$$= \frac{S}{2} \frac{\lambda_2}{\lambda_1+\lambda_2} \frac{\lambda_1}{\lambda_1+\mu_2} + \left(\frac{1}{\mu_1} + \frac{S}{2} \right) \frac{\lambda_1}{\lambda_1+\lambda_2} \frac{\lambda_2}{\lambda_2+\mu_1} \qquad (6.11)$$

Note that under the assumptions of our deadlock model listed at the beginning of this section, a read only transaction (i.e. a transaction with an empty writeset) will not enter into a deadlock with another transaction. Consider two transactions T_1 and T_2, in which T_1 has non-empty readset and writeset and T_2 has empty writeset. Suppose T_1 arrives first and queues at all files in its readset. After a certain time T_2 arrives. Since two read requests do not conflict, T_2 can be completed without having to wait for T_1 to complete. Suppose T_2 arrives first. Again T_2 does not have to wait for T_1. Therefore, no deadlock is possible.

A write only transaction (i.e. a transaction with empty readset) may eneter into a deadlock with another transaction. Consider two transactions T_1 and T_2, in which T_1 has readset (X) and writeset (Y) and T_2 has an empty readset, and writeset (X,Y). Suppose T_1 arrives first and locks X. After a time T_2 arrives, it joins the queue for file X and locks file Y. When T_1 performs the write step, it cannot access file Y and must wait for T_2. A deadlock is thereby created. On the other hand, if T_2 arrives first, then it does not have to wait for T_1 and no deadlock is possible.

6.3 Timestamp Ordering (SDD-1)

The conflict model of SDD-1 attempts to determine two important parameters : (1)p_α, the possibility that a read message will be rejected because of an obsolete (or reversed) timestamp, and (2) given that a read message is not rejected, the time W_α it has to wait before the read condition is satisfied and it can be processed.

6.3.1 Probability of Read Rejection

Consider the simplest case of two TM's shown in Fig. 6.9. Transactions i_α arrive at TM_α with Poisson rate λ_α while conflicting transactions i_β arrive at TM_β with rate λ_β. Assume that, upon reaching TM_α and TM_β, i_α and i_β take respectively time t_α and t_β to get to DM_α. These times include both the queueing and transmission times at the respective channels. Suppose we choose an arbitrary i_α, and consider the time we have to wait until we see the next arrival of an i_β at TM_β. Call this waiting time a_β. Due to the memoryless property of Poisson processes, we note that a_β is exponential with rate λ_β. An inspection of Fig. 6.9(b) gives us the following expression for p_α:

p_α = P(i_α will arrive at DM_α later than i_β given that i_α enters the database system at TM_α before i_β enters the system at TM_β)

$= P(t_\alpha > a_\beta + t_\beta)$

$= P(a_\beta < t_\alpha - t_\beta)$

Case 1: Suppose that t_α and t_β are constants,

$$\text{then } p_\alpha = \begin{cases} 1 - e^{-\lambda_\beta (t_\alpha - t_\beta)} & \text{if } t_\alpha > t_\beta \\ 0 & \text{otherwise} \end{cases}$$

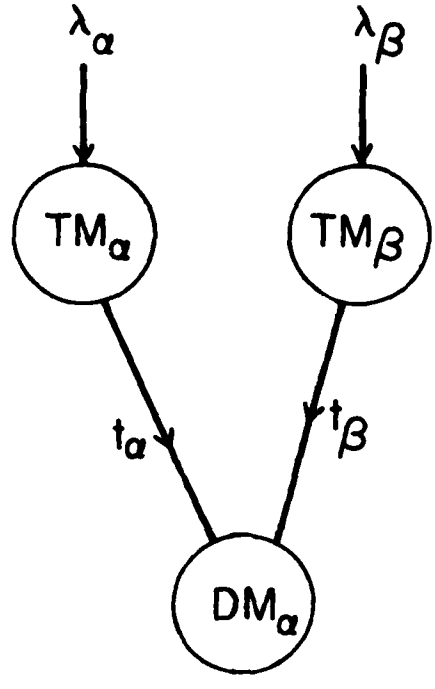
Case 2: the lengths of messages i_α and i_β are exponentially distributed with mean $1/\mu_\alpha$ and $1/\mu_\beta$.

In this case, the analysis of Case 1 is still applicable, except that $t_\alpha - t_\beta$ is not a constant anymore.

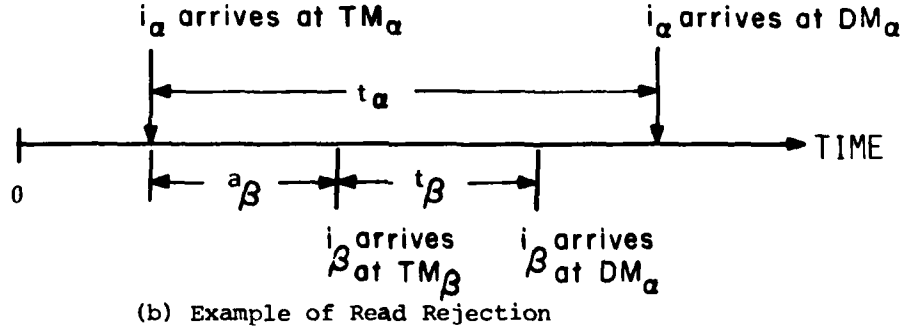
Recall that if y is the total service time (queueing plus service) for an M/M/1 queue, then

$$f_y(y_0) = (\mu - \lambda)\, e^{-(\mu - \lambda) y_0} \qquad y_0 \geq 0$$

where λ is the arrival rate and μ is the service rate. Therefore t_α and t_β



(a) Conflicting Transactions arriving at DM_α



(b) Example of Read Rejection

Figure 6.9 Probability of Read Rejection

are exponential with rates $\mu_\alpha-\lambda_\alpha$ and $\mu_\beta-\lambda_\beta$ respectively. Moreover, the pdf of $T = t_\alpha - t_\beta$, given that $t_\alpha > t_\beta$ will be given by

$$f_T(T_o) = (\mu_\alpha-\lambda_\alpha)\, e^{-(\mu_\alpha-\lambda_\alpha)T_o} \qquad T_o \geq 0$$

due to the memoryless property of Poisson processes.

Therefore, $p_\alpha = P(t_\alpha > t_\beta)\ P(t_\alpha - t_\beta > a_\beta \mid t_\alpha > t_\beta)$

$$= \frac{\mu_\beta-\lambda_\beta}{\mu_\beta-\lambda_\beta+\mu_\alpha-\lambda_\alpha}\ \frac{\lambda_\beta}{\lambda_\beta+\mu_\alpha-\lambda_\alpha}$$

Case 3: General Case - More than two TM's, messages with exponential lengths.

Let $G = \{\beta,\gamma,\ldots\}$ be the set of subscripts of those TM's that send messages to DM_α which conflict with message i_α. Consider the TM pairs $(TM_\alpha, TM_\beta), (TM_\alpha, TM_\gamma), \ldots$ in a similar fashion to the analysis in Case 2. The probability of rejection due to each TM_g, $g\varepsilon G$, can be found as above. Since the arrival of messages at the TM's are independent,

$$\begin{aligned} p_\alpha &= P(i_\alpha \text{ will be read rejected}) \\ &= 1 - P(i_\alpha \text{ will not be read rejected}) \\ &= 1 - \prod_{g\varepsilon G} P(i_\alpha \text{ will not be read rejected by messages from } TM_g) \\ &= 1 - \prod_{g\varepsilon G} \left[\, 1 - \frac{\mu_g-\lambda_g}{\mu_g-\lambda_g+\mu_\alpha-\lambda_\alpha}\ \frac{\lambda_g}{\lambda_g+\mu_\alpha-\lambda_\alpha}\right] \end{aligned}$$

6.3.2 Delay due to conflicts

What is the read delay given that a read message is not rejected?

Case 1: Two TM's, exponential message lengths.

An inspection of Fig. 6.10 gives

$$W_\alpha = (a_\beta + t_\beta - t_\alpha \mid a_\beta + t_\beta \geq t_\alpha)$$

$$= \begin{cases} (a_\beta - t_\alpha) + t_\beta & \text{if } a_\beta \geq t_\alpha \\ t_\beta - (t_\alpha - a_\beta) & \text{if } a_\beta \leq t_\alpha \leq a_\beta + t_\beta \end{cases}$$

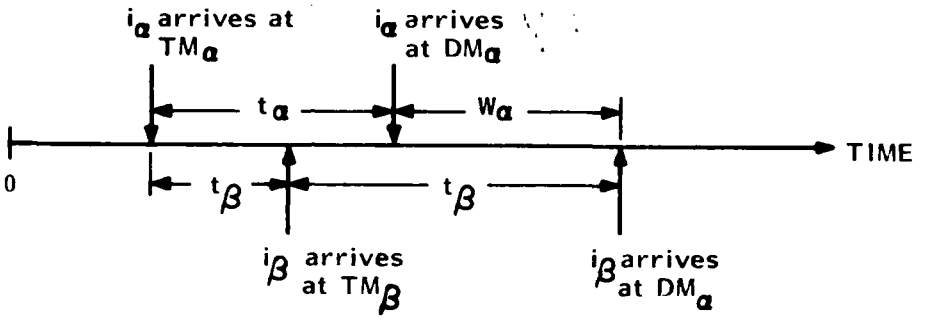


Figure 6.10 Finding W_α, the Delay due to Conflicting Transactions

$$\text{Therefore, } W_\alpha \sim \begin{cases} a_\beta + t_\beta & \text{with probability } (\mu_\alpha-\lambda_\alpha)/(\lambda_\beta+\mu_\alpha-\lambda_\alpha) \\ t_\beta & \text{with prob. } \lambda_\beta/(\lambda_\beta+\mu_\alpha-\lambda_\alpha) \end{cases} \quad (6.13)$$

since $a_\beta - t_\beta \sim a_\beta$ if $a_\beta > t_\beta$, and $t_\beta - (t_\alpha - a_\beta) \sim t_\beta$ if $t_\alpha - a_\beta \geq 0$ and $t_\beta \geq t_\alpha - a_\beta$.

$$\text{Hence, } f_{W_\alpha}(x) = \frac{\mu_\alpha-\lambda_\alpha}{\lambda_\beta+\mu_\alpha-\lambda_\alpha} f_u(x) + \frac{\lambda_\beta}{\lambda_\beta+\mu_\alpha-\lambda_\alpha} f_{t_\beta}(x) \qquad x \geq 0$$

where $u = a_\beta + t_\beta$.

$$\begin{aligned} E(W_\alpha) &= \frac{\mu_\alpha-\lambda_\alpha}{\lambda_\beta+\mu_\alpha-\lambda_\alpha} E(a_\beta + t_\beta) + \frac{\lambda_\beta}{\lambda_\beta+\mu_\alpha-\lambda_\alpha} E(t_\beta) \\ &= E(t_\beta) + \frac{\mu_\alpha-\lambda_\alpha}{\lambda_\beta+\mu_\alpha-\lambda_\alpha} E(a_\beta) \\ &= \frac{1}{\mu_\beta-\lambda_\beta} + \frac{\mu_\alpha-\lambda_\alpha}{\lambda_\beta+\mu_\alpha-\lambda_\alpha} \cdot \frac{1}{\lambda_\beta} \end{aligned} \quad (6.14)$$

Case 2: If we have more than two TM's, then we must wait until the arrival of all conflicting writes with b gger timestamp than i_α.

$$\text{Therefore, } W_\alpha = \max_{g \in G} \left(a_g + t_g - t_\alpha \,|\, a_g + t_g \geq t_\alpha \right)$$

where $G = \{\beta, \gamma, \ldots\}$ = set of subscripts of all TM's that send messages conflicting with i_α to DM_α, a_g is the interarrival time of messages at TM_g and t_g is the time these messages take to get to DM_α.

$$\text{Let } x_g = (a_g + t_g - t_\alpha \,|\, a_g + t_g \geq t_\alpha), \text{ then } W_\alpha = \max_{g \in G} (x_g)$$

Let $v = a_g + t_g$. Since a_g and t_g are independent,

$$f_v(x) = \frac{\lambda_1\lambda_2}{\lambda_2 - \lambda_1} e^{-\lambda_1 x} + \frac{\lambda_1\lambda_2}{\lambda_1 - \lambda_2} e^{-\lambda_2 x} \qquad x \geq 0$$

where $\lambda_1 = \lambda_g$, and $\lambda_2 = \mu_g - \lambda_g$.

$$\text{Now, } x_g \sim \begin{cases} a_g + t_g & \text{with probability } (\mu_\alpha-\lambda_\alpha)/(\lambda_g+\mu_\alpha-\lambda_\alpha) \\ t_g & \text{with probability } \lambda_g/(\lambda_g+\mu_\alpha-\lambda_\alpha) \end{cases}$$

(See Equ. 6.13))

$$\text{Therefore, } f_{x_g}(x) = \frac{\mu_\alpha - \lambda_\alpha}{\lambda_g + \mu_\alpha - \lambda_\alpha} \left(\frac{\lambda_1 \lambda_2}{\lambda_2 - \lambda_1} e^{-\lambda_1 x} + \frac{\lambda_1 \lambda_2}{\lambda_1 - \lambda_2} e^{-\lambda_2 x} \right)$$

$$+ \frac{\lambda_g}{\lambda_g + \mu_\alpha - \lambda_\alpha} \lambda_2 e^{-\lambda_2 x} \qquad x \geq 0$$

$$F_{x_g}(x) = P(x_g \leq x) = \frac{\mu_\alpha - \lambda_\alpha}{\lambda_g + \mu_\alpha - \lambda_\alpha} \frac{\lambda_2}{\lambda_2 - \lambda_1} \left(1 - e^{-\lambda_1 x} \right)$$

$$+ \left(\frac{\mu_\alpha - \lambda_\alpha}{\lambda_g + \mu_\alpha - \lambda_\alpha} \frac{\lambda_1}{\lambda_1 - \lambda_2} + \frac{\lambda_g}{\lambda_g + \mu_\alpha - \lambda_\alpha} \right) \left(1 - e^{-\lambda_2 x} \right) \qquad x \geq 0$$

$$\text{Now, } F_{W_\alpha}(x) = P(W_\alpha \leq x)$$

$$= P(\max. x_g \leq x)$$

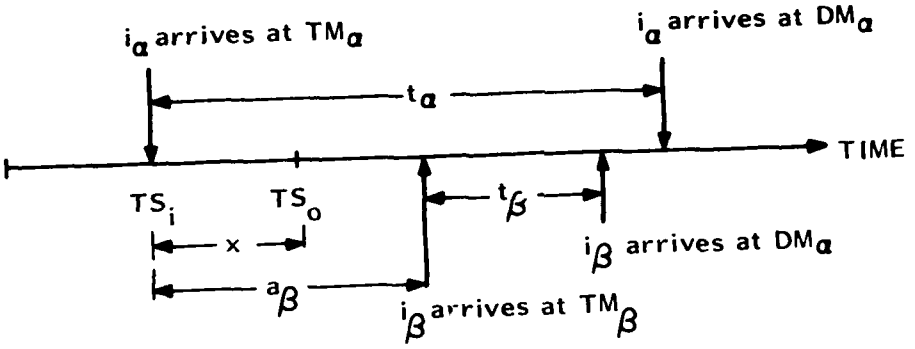
$$= \prod_{g \in G} (x_g \leq x)$$

$$\text{Hence, } f_{W_\alpha}(x) = \frac{d}{dx} F_{W_\alpha}(x), \text{ and } E(W_\alpha) = \int_0^\infty (1 - F_{W_\alpha}(x))\, dx.$$

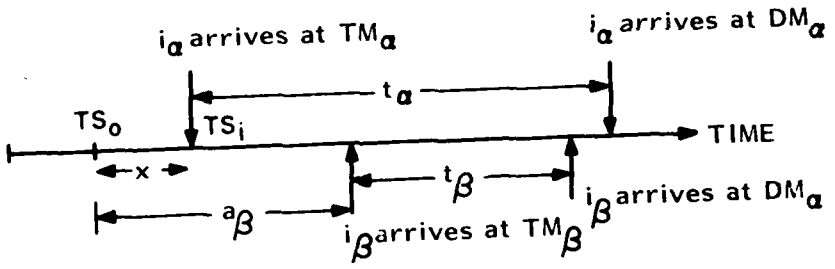
Since $F_{x_g}(x)$ is known for all $g \in G$, the last two expressions can be evaluated.

6.3.3 Optimal Read Conditions

In deriving p_α and W_α in the last two sections, we have assumed that the timestamp in the read condition, call it TS_o, is the same as the time when the transaction arrives at TM_α (TS_i). Note that this is true only for transactions running under protocol P3. For messages running under protocols P1 and P2, the timestamp in the read condition can be chosen arbitrarily by the TM, the only requirement being that all RR messages sent from the same TM on behalf of a transaction have the same read condition. In particular, TM_α can choose $TS_o = TS_i + x$, or $TS_o = TS_i - x$, where x is an arbitrary constant. In [BSR80], the builders of SDD-1 point out that the choice of this read condition timestamp is an important parameter that must be finetuned since the efficiency of the system depends on it. Too



(a) Case 1: $TS_o = TS_i + x$



(b) Case 2: $TS_o = TS_i - x$

Figure 6.11 Probability of Rejection and Delay due to Conflicts given Arbitrary Read Conditions

small a read condition timestamp will lead to a lot of RR messages being read rejected, while too large a read condition timestamp will incur an excessive W_α, the waiting time until a conflicting write message with timestamp greater than TS_o arrives. We now formulate an optimization problem to determine the best value of TS_o under various situations. This approach will probably be better than trying to arrive at a good TS_o by trial and error. We shall consider the case of two conflicting TM's only. The result can be extended to the general case of many conflicting TM's, in a fashion similar to that described in the last section.

Case 1: $TS_o = TS_i + x$

Inspection of Fig. 6.11(a) gives

$p_\alpha = P(i_\alpha$ arrives at DM_α after $i_\beta \mid i_\beta$ has timestamp greater than $TS_o)$

$= P(t_\alpha > a_\beta + t_\beta \mid a_\beta \geq x)$

$W_\alpha = (a_\beta + t_\beta - t_\alpha \mid a_\beta \geq x,\ a_\beta + t_\beta \geq t_\alpha)$

$a_\beta \geq x$ assures that the timestamp of $i_\beta \geq TS_o$.

p_α can be rewritten as:

$p_\alpha = P(a_\beta < T \mid a_\beta \geq x)$ where $T = t_\alpha - t_\beta$

a_β is the remaining time one has to wait until the arrival of i_β at TM_β, given that one enters the system at a random time. For a Poisson process, this remaining time due to random incidence is still exponential with the same parameter λ_β.

Therefore, $f_{a_\beta \mid a_\beta \geq x}(a \mid a \geq x) = \lambda_\beta e^{-\lambda_\beta (a-x)} \qquad a \geq x$

For $T \geq x$, $p_\alpha = \int_x^T \lambda_\beta e^{-\lambda_\beta (a-x)}\, da = 1 - e^{-\lambda_\beta T} e^{\lambda_\beta x}$ (6.15)

For $T < x$, $p_\alpha = 0$

Since T is a random variable, we have to integrate Equ.(6.15) over the pdf of T. Now

$$f_{T|T\geq x}(T_o \mid T_o \geq x) = \mu_1 e^{-\mu_1(T_o - x)} \qquad T_o \geq x$$

where $\mu_1 = \mu_\alpha - \lambda_\alpha$

Therefore,
$$p_\alpha = \int_x^\infty (1 - e^{-\lambda_\beta T} e^{\lambda_\beta x}) \mu_1 e^{-\mu_1(T-x)} dT$$
$$= \mu_1 \left[e^{\mu_1 x} \cdot \frac{e^{-\mu_1 T}}{-\mu_1} - e^{(\lambda_\beta + \mu_1)x} \cdot \frac{e^{-(\lambda_\beta+\mu)T}}{-(\lambda_\beta+\mu_1)} \right]_x^\infty$$
$$= \begin{cases} \lambda_\beta/(\lambda_\beta + \mu_1) & \text{for } T \geq x \\ 0 & \text{for } T < x \end{cases} \qquad (6.16)$$

We next determine $P(T \geq x) = P(t_\alpha \geq t_\beta + x)$.

Consider the joint pdf space of t_α and t_β (Fig. 6.12).

Let $\mu_1 = \mu_\alpha - \lambda_\alpha$, $\mu_2 = \mu_\beta - \lambda_\beta$, then

$P(t_\alpha \geq t_\beta + x)$ = shaded area

$$= \int_{t_\alpha = x}^\infty dt_\alpha \int_{t_\beta = 0}^{t_\alpha - x} dt_\beta \, \mu_1 \mu_2 e^{-\mu_1 t_\alpha} e^{-\mu_2 t_\beta}$$
$$= \mu_1 \mu_2 \int_{t_\alpha = x}^\infty dt_\alpha \, e^{-\mu_1 t_\alpha} \left[\frac{e^{-\mu_2 t_\beta}}{-\mu_2} \right]_{t_\beta = 0}^{t_\alpha - x}$$
$$= \mu_1 \int_{t_\alpha = x}^\infty (e^{-\mu_1 t_\alpha} - e^{\mu_2 x} e^{-(\mu_1 + \mu_2) t_\alpha}) dt_\alpha$$
$$= \mu_2 e^{-\mu_1 x}/(\mu_1 + \mu_2)$$

From Equ.(6.15),
$$p_\alpha = \frac{\lambda_\beta}{\lambda_\beta + \mu_1} \cdot P(T \geq x),$$

therefore,
$$p_\alpha = \frac{\lambda_\beta}{\lambda_\beta + \mu_\alpha - \lambda_\alpha} \cdot \frac{(\mu_\beta - \lambda_\beta) e^{-(\mu_\alpha - \lambda_\alpha)x}}{\mu_\beta - \lambda_\beta + \mu_\alpha - \mu_\alpha} \qquad (6.17)$$

Equ.(6.17) says that P_α decreases with increasing values of x, which is in tuitively correct.

We next calculate W_α, as follows:

$$W_\alpha = (a_\beta + t_\beta - t_\alpha \mid a_\beta + t_\beta \geq t_\alpha, a_\beta \geq x)$$
$$= \begin{cases} (a_\beta - t_\alpha) + t_\beta & \text{if } a_\beta \geq t_\alpha, \text{ given } a_\beta \geq x \\ t_\beta - (t_\alpha - a_\beta) & \text{if } a_\beta \leq t_\alpha \leq a_\beta + t_\beta, \text{ given } a_\beta \geq x \end{cases}$$

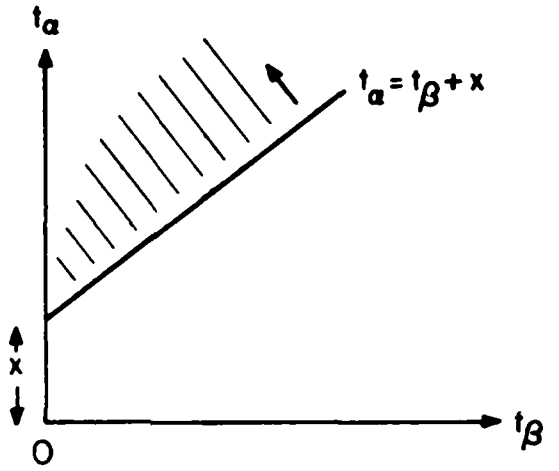


Figure 6.12 Probability $t_\alpha \geq t_\beta$

$$W_\alpha \sim \begin{cases} a_\beta + t_\beta & \text{if } a_\beta \geq t_\alpha \text{, given } a_\beta \geq x \\ t_\beta & \text{if } a_\beta < t_\alpha \text{, given } a_\beta \geq x \end{cases}$$

$$\begin{aligned} E(W_\alpha) &= E(a_\beta + t_\beta \mid a_\beta \geq x) \cdot P(a_\beta \geq t_\alpha \mid a_\beta \geq x) \\ &\quad + E(t_\beta \mid a_\beta \geq x) \cdot P(a_\beta < t_\alpha \mid a_\beta \geq x) \\ &= E(t_\beta) + E(a_\beta \mid a_\beta \geq x) \cdot P(a_\beta \geq t_\alpha \mid a_\beta \geq x) \end{aligned}$$

Now, $$P(a_\beta \geq t_\alpha \mid a_\beta \geq x) = \begin{cases} \mu_1/(\lambda_\beta + \mu_1) & \text{if } t_\alpha \geq x \\ 1 & \text{if } t_\alpha < x \end{cases}$$

$$= \mu_1 e^{-\mu_1 x}/(\lambda_\beta + \mu_1) + 1 - e^{-\mu_1 x}$$

Therefore, $$E(W_\alpha) = 1/\mu_2 + \left(\frac{1}{\lambda_\beta} + x\right)\left(1 - \frac{\lambda_\beta e^{-\mu_1 x}}{\lambda_\beta + \mu_1}\right) \qquad (6.18)$$

Case 2: $TS_0 = TS_i - x$

Inspection of Fig. 6.11(b) gives

$p_\alpha = P(t_\alpha + x \geq a_\beta + t_\beta)$

$W_\alpha = (a_\beta + t_\beta - t_\alpha - x \mid a_\beta + t_\beta \geq t_\alpha + x)$

Note that any i_β that arrives at the system at a time after TS_o will have a timestamp bigger than the read condition of i_α. Moreover, due to the memoryless property of Poisson processes, a_β is still exponential with mean $1/\lambda_\beta$.

p_α can be rewritten as: $$p_\alpha = P(a_\beta \leq T + x) \quad \text{where } T = t_\alpha - t_\beta$$

$$= \begin{cases} 1 - e^{-\lambda_\beta(T+x)} & \text{for } T + x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Now T, being the difference of two exponentials, has a pdf as shown in Fig. 6.13.

Hence, $$E(e^{-\lambda_\beta T}) = \frac{\mu_1 \mu_2}{\mu_1 + \mu_2}\left[\int_{-x}^{0} e^{-\lambda_\beta T_o} e^{\mu_2 T_o} d T_o + \int_{0}^{\infty} e^{-\lambda_\beta T_o} e^{-\mu_1 T_o} d T_o\right]$$

$$= \frac{\mu_1 \mu_2}{\mu_1 + \mu_2}\left[\frac{1 - e^{-(\mu_2 - \lambda_\beta)x}}{\mu_2 - \lambda_\beta} + \frac{1}{\lambda_\beta + \mu_1}\right] \qquad (6.19)$$

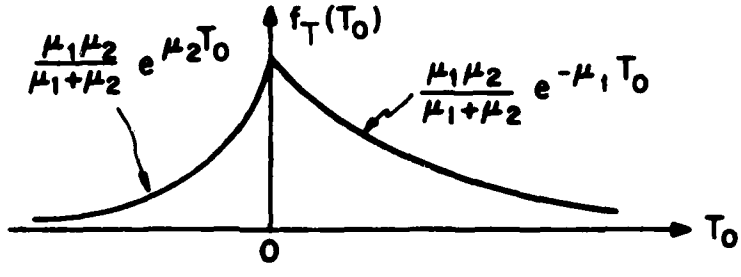


Figure 6.13 pdf of T, the difference of two exponentials

$$\text{and, } P(T+x \geq 0) = 1 - P(T+x < 0)$$
$$= 1 - P(T < -x)$$
$$= 1 - \int_{-\infty}^{-x} \frac{\mu_1 \mu_2}{\mu_1 + \mu_2} e^{\mu_2 T} dT$$
$$= 1 - \mu_1 e^{-\mu_2 x}/(\mu_1 + \mu_2) \qquad (6.20)$$

Hence $p_\alpha = [1 - e^{-\lambda_\beta x} E(e^{-\lambda_\beta T})] P(T+x \geq 0)$ (6.21)

where $E(e^{-\lambda_\beta T})$ and $P(T+x \geq 0)$ are given by (6.19) and (6.20)

$$W_\alpha = (t_\beta + a_\beta - t_\alpha - x \mid t_\beta + a_\beta \geq t_\alpha + x)$$
$$= \begin{cases} a_\beta - (t_\alpha + x) + t_\beta & \text{if } a_\beta \geq t_\alpha + x \\ t_\beta - (t_\alpha + x - a_\beta) & \text{if } a_\beta \leq t_\alpha + x \leq a_\beta + t_\beta \end{cases}$$

$$W_\alpha \sim \begin{cases} a_\beta + t_\beta & \text{if } a_\beta \geq t_\alpha + x \\ t_\beta & \text{if } a_\beta \leq t_\alpha + x \leq a_\beta + t_\beta \end{cases}$$

$$E(W_\alpha) = E(t_\beta) + E(a_\beta) \cdot P(a_\beta \geq t_\alpha + x)$$
$$= E(t_\beta) + E(a_\beta) \cdot P(y \geq x) \quad \text{where } y = a_\beta - t_\alpha$$

$$\text{Now, } f_y(y_0) = \begin{cases} \mu_1 \lambda_\beta e^{\mu_1 y}/(\mu_1 + \lambda_\beta) & \text{for } y < 0 \\ \mu_1 \lambda_\beta e^{-\lambda_\beta y_0}/(\mu_1 + \lambda_\beta) & \text{for } y \geq 0 \end{cases}$$

$$\text{Hence, } P(y \geq x) = \int_x^\infty \frac{\mu_1 \lambda_\beta}{\mu_1 + \lambda_\beta} e^{-\lambda_\beta y_0} dy_0$$
$$= \mu_1 e^{-\lambda_\beta x}/(\mu_1 + \lambda_\beta)$$

$$E(W_\alpha) = 1/\mu_2 + \mu_1 e^{-\lambda_\beta x}/\lambda_\beta(\mu_1 + \lambda_\beta) \qquad (6.22)$$

where $\mu_1 = \mu_\alpha - \lambda_\alpha$, $\mu_2 = \mu_\beta - \lambda_\beta$.

For both Case 1 and Case 2,

$$E(\text{delay of a random read message}) = p_\alpha [D + E(W_\alpha)] + (1 - p_\alpha) E(W_\alpha) \qquad (6.23)$$

where D is the penalty due to a read rejection. The expected delay can thus be minimized, for given values of $D, \lambda_\alpha, \lambda_\beta, \mu_\alpha, \mu_\beta$, by varying x.

6.3.4 Optimal Time to Send Nullwrites

A read message with read condition $\langle TS_o, (\bar{j}, \bar{k}, \ldots) \rangle$ arriving at DM_α must wait for the arrival of write messages from conflicting classes $(\bar{j}, \bar{k} \ldots)$ with timestamp greater than TS_o. If no conflicting write message comes into the system, there will be an excessive wait. One proposal to remedy this problem is to have TM's send nullwrite messages periodically.

Consider a database system in which nullwrites will be sent whenever the time since the last write message is greater than S. The interarrival time of write message (both regular writes and nullwrites) will now be

$$f_{a_\beta}(a) = \lambda_\beta e^{-\lambda_\beta a} \cdot \frac{1}{1 - e^{-\lambda_\beta S}} \qquad 0 \le a \le S \qquad (6.24)$$

ie. an exponential constrained to be less than S.

Consider an instant of time when a read message has just arrived at TM_α. The time a_β', we have to wait until the arrival of a conflicting write message at TM_β is not of the form given in (6.24), but rather has a pdf given by random incidence equations (See [DRAK67])

$$f_{a_\beta'}(a) = \frac{1 - P(a_\beta \le a)}{E(a_\beta)}$$

$$\text{Now,} \quad P(a_\beta \le a) = \begin{cases} \int_o^a \lambda_\beta e^{-\lambda_\beta a} \frac{1}{1 - e^{-\lambda_\beta S}} \, d a_\beta \\ = (1 - e^{-\lambda_\beta a}) / (1 - e^{-\lambda_\beta S}) & a \le S \\ 1 & a > S \end{cases}$$

$$\text{and,} \quad E(a_\beta) = \int_o^S a_\beta \lambda_\beta e^{-\lambda_\beta a_\beta} \cdot \frac{1}{1 - e^{-\lambda_\beta S}} \, d a_\beta$$

$$= \frac{1 - e^{-\lambda_\beta S} (\lambda_\beta S + 1)}{\lambda_\beta (1 - e^{-\lambda_\beta S})}$$

$$\text{Hence,} \quad f_{a_\beta}(a) = \frac{\lambda_\beta (e^{-\lambda_\beta a} - e^{-\lambda_\beta S})}{1 - e^{-\lambda_\beta S} (\lambda_\beta S + 1)} \qquad 0 \le a \le S \qquad (6.25)$$

By inspection of Fig. 6.9(b) and Fig. 6.10, in which a_β must be replaced by a_β', we get:

$$p_\alpha = P(t_\alpha > a_\beta' + t_\beta)$$

$$W_\alpha = (a_\beta' + t_\beta - t_\alpha \mid a_\beta' + t_\beta > t_\alpha)$$

Now, $P(t_\alpha > a_\beta' + t_\beta) = P(t_\alpha > a_\beta')\, P(t_\alpha - a_\beta' > t_\beta \mid t_\alpha > a_\beta')$

$$= P(t_\alpha > a_\beta')\, P(t_\alpha > t_\beta) \qquad (6.26)$$

Let $A = 1 - e^{-\lambda_\beta S}(\lambda_\beta S + 1)$

$$P(a_\beta' \geq t_\alpha) = \int_{a=0}^{S} \int_{t=0}^{a} \mu_1 e^{-\mu_1 t} \lambda_\beta (e^{-\lambda_\beta a} - e^{-\lambda_\beta S})/A \, dt\, da$$

$$= 1 - B \qquad (6.27)$$

where $B = \frac{\lambda_\beta}{A} \cdot \frac{\mu_1 - e^{-\lambda_\beta S}(\mu_1 + \lambda_\beta - \lambda_\beta e^{-\mu_1 S})}{\mu_1(\mu_1 + \lambda_\beta)}$

$P(a_\beta' < t_\alpha) = 1 - P(a_\beta' \geq t_\alpha)$ and $P(t_\alpha > t_\beta) = \mu_2/(\mu_1 + \mu_2)$

Hence, from Equ. (6.26), $p_\alpha = \mu_2 B/(\mu_1 + \mu_2)$ (6.28)

$$W_\alpha = (a_\beta' + t_\beta - t_\alpha \mid a_\beta' + t_\beta \geq t_\alpha)$$

$$E(W_\alpha) = E(t_\beta) + E(a_\beta')\, P(a_\beta' \geq t_\alpha)$$

Now, $E(a_\beta') = \int_0^S a \lambda_\beta (e^{-\lambda_\beta a} - e^{-\lambda_\beta S})/A \, da$

$$= 1/\lambda_\beta - \lambda_\beta S^2 e^{-\lambda_\beta S}/2A$$

$$E(W_\alpha) = \frac{1}{\mu_2} + \left(\frac{1}{\lambda_\beta} - \frac{\lambda_\beta S^2 e^{-\lambda_\beta S}}{2A}\right)(1 - B) \qquad (6.29)$$

Substituting these values of p_α and $E(W_\alpha)$ into Equ.(6.23) for the expected delay of a random read message, we can minimize the expected delay by choosing an appropriate value for S.

In Appendix III, we have developed expressions for p_α and $E(W_\alpha)$ when there are more than two conflicting transaction classes.

We can also derive expressions for p_α and $E(W_\alpha)$ for cases where $TS_o = TS_i + x$ and $TS_o = TS_i - x$, as we did in section 6.3.3. However, for

exponential interarrival time constrained to be less than S, these expressions become extremely complex. Therefore, they are not included in this thesis.

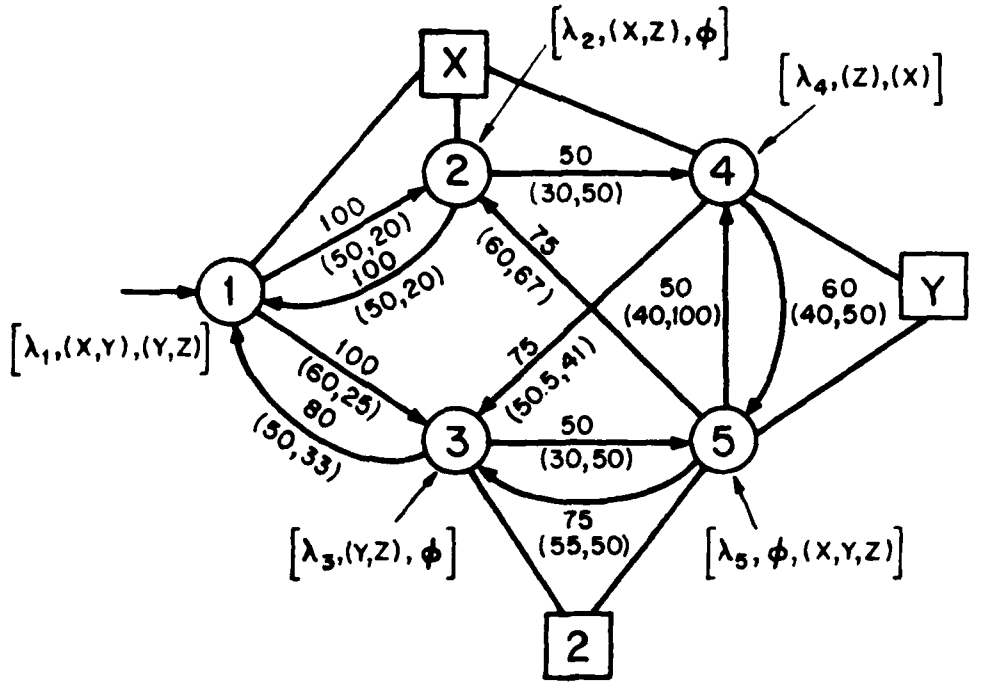
CHAPTER 7

NUMERICAL EXAMPLES

We shall now demonstrate how to model various concurrency control algorithms. Four examples will be given: (1) Centralized Locking Algorithm with Deadlock Detection, (2) Distributed Locking Algorithm with Ordered Queues for Deadlock Prevention, (3) Distributed Locking with Prioritized Transactions for Deadlock Prevention, and (4) SDD-1.

Fig. 7.1 shows the example distributed database that we are going to use this chapter. It consists of a communication subnetwork with five nodes. For each communication channel, all of which are directed, we have indicated its capacity, the existing message flow (i.e. not counting anticipated distributed database traffic) and the existing transmission delay. There are three files X, Y and Z with redundant copies. The location of the redundant copies are as indicated by the artificial file node and the artificial links. There are also five classes of transactions and their arrival rates are as indicated.

Recall that our DDB Model consists of five steps. The input data contained in Fig.7.1 will be collected in Step 1. Step 2 is the Transaction Processing Model, which consists of the Query Processing Step and the Write Step. Using existing delay figures on the communication channels as input to the MST1 Query Processing Algorithm, we found which nodes a particular transaction will access to read a file. For example, class 1 transactions, with readset (X,Y), will read file X at node 1 and file Y at node 4. The transactions must also write on all copies of files in their writesets. The nodes accessed by different transactions for reads and writes are summarized in Fig.7.2. Using this information, we can estimate the additional traffic generated on



COMPUTER SITE

FILE NODE

100 — capacity in Kbs

communication channel

(50,20) — present delay in msec

existing flow in Kbits

$[\lambda_1,(X,Y),(Y,Z)]$

writeset

readset

arrival rate of transaction class 1

Figure 7.1 Example Distributed Database

Transaction i \ Node j	1	2	3	4	5
1	reads X		writes Z	reads Y writes Y	writes Y,Z
2		reads X	reads Z		
3			reads Z	reads Y	
4	writes X	writes X		writes X	reads Z
5	writes X	writes X	writes Z	writes X,y	writes Y,Z

Transaction i reads (or writes) files at node j.

Figure 7.2 Nodes Acessed by Different Classes of Transactions according to the Transaction Processing Model

each communication channel by a particular transaction under different concurrency control algorithms.

In Step 3 we calculate the transmission delay at each communication channel, given the additional database traffic. The additional traffic and hence the resultant delays on the channels will be different under different concurrency control algorithms. This will be discussed in more detail for each example. We distinguish between short messages with average length $1/\mu_1$ = .1 Kbit and long messages with mean length of $1/\mu_2$ = 1 Kbit. Short messages include lock requests, lock releases, lock grants, read requests, commits, and acknowledgement messages. Long messages include file transfers and pre-commits. Each communication channel is modelled as an M/M/1 FCFS queue with mean service time = $1/\mu$ = $\frac{1}{\gamma_1+\gamma_2}(\frac{\gamma_1}{\mu_1} + \frac{\gamma_2}{\mu_2})$ where γ_1, γ_2 are the arrival rates of the short and long messages respectively.

In Step 4, we estimate the probability of conflicts and the delay due to conflict. Conflict models for the four concurrency control algorithms were developed in Chapter 6.

In Step 5 we calculate the response time which is a sum of the query processing delay, the write delay and the delay due to conflicts.

7.1 Centralized Two-Phase Locking

Suppose Computer site 1 is chosen as the central node. (See Fig.7.1). All transactions have to request locks from this node.

We now consider the message flow generated by the DDB management system on behalf of the transactions. Fig.7.3 summarizes the sequence of events corresponding to the processing of each transaction under

Centralized Two-Phase Locking:

(1) transaction sends requests to central node to lock files in the readset,

(2) wait for read locks at central node,

(3) central node sends lock grant message to request node,

(4) query processing, result produced at request node,

(5) transaction sends request to central node to lock files in the writeset,

(6) wait for write locks at central node,

(7) central node sends lock grant message to request node,

(8) request node sends pre-commit messages to copies of files written on,

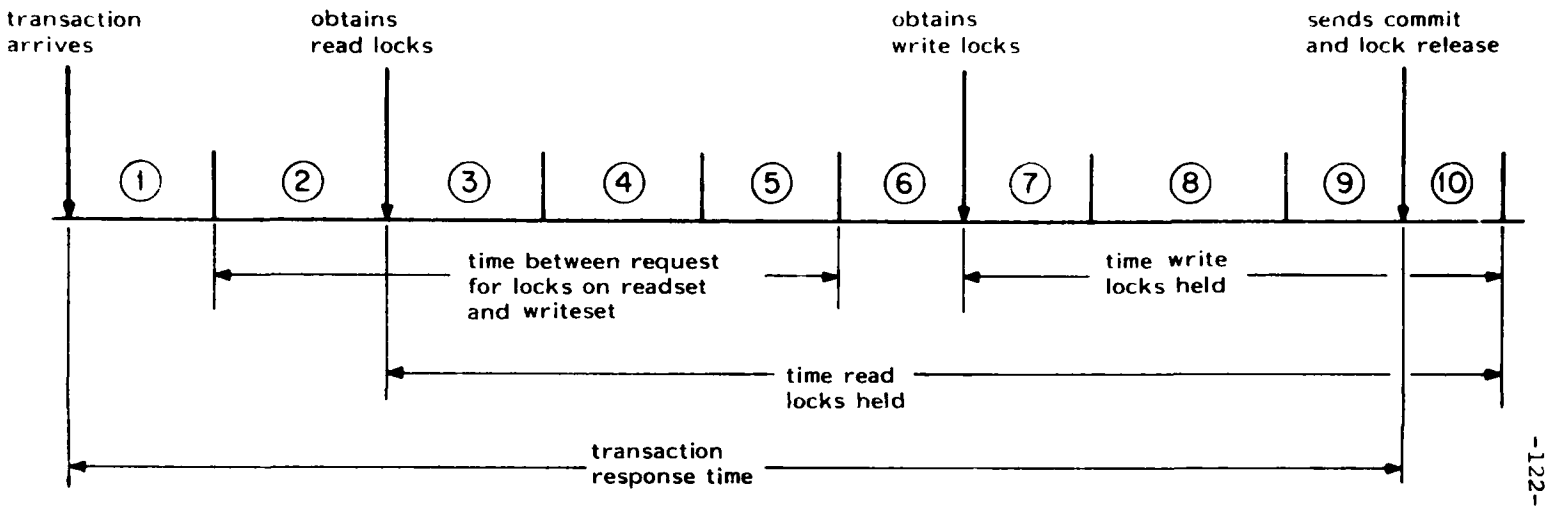
(9) copies send acknowledgement messages to request node,

(10) request node sends commit messages to copies and lock release message to central node.

For example, consider transactions arriving at node 4.

Message description	channels traversed	message type
λ_4 read lock request from nodes 4 to 1	C_{43}, C_{31}	short
λ_4 read lock grant from 1 to 4	C_{12}, C_{24}	short
query processing: read request to 5	C_{45}	short
file Z transferred from 5 to 4	C_{54}	long
λ_4 write lock request from 4 to 1	C_{43}, C_{31}	short
λ_4 write lock grant from 1 to 4	C_{12}, C_{24}	short
λ_4 pre-commit messages to all copies of X	C_{43}, C_{31}, C_{12}	long
λ_4 acknowledgement messages	C_{12}, C_{24}	short
λ_4 commit messages to all copies of X	C_{43}, C_{31}, C_{12}	short
λ_4 lock release from 4 to 1	C_{43}, C_{31}	short

Similar considerations for the other transactions give the additional



Key:
① req. node sends read lock req. to central node
② queues for read locks at central node
③ central node sends lock grant to req. node
④ query processing, result produced at req. node
⑤ req. node sends write lock req. to central node
⑥ queues for write lock at central node
⑦ central node sends lock grant to req. node
⑧ req. node sends pre-commits to copies written on
⑨ copies send acknowledgement to req. node
⑩ req. node sends lock release to central node

Figure 7.3 Chronological Events corresponding to Transaction Processing under Centralized Two-phase Locking

message flow requirements generated by the database management system. We can then calculate the expected message delay on each of the communication channels. For example, for channel C_{13}, we have $\lambda_1 + \lambda_2 + \lambda_3 + \lambda_5 = .8$ additional short messages per second, and $\lambda_1 = .3$ additional long messages per second. Assuming that the existing message traffic of 60 kbit per second on C_{13}, i.e. not including the DDB traffic, are all long messages, then the total number of messages on channel C_{21} is 60 + .8 + .3 + 61.1 message per second, with an average message length of (60.3 ×1K + .8 ×.1K)/61.1 = .988 Kbits. The expected queueing delay* = 61.1/(.988 K(100) - 61.1) × (.988 K(100)) = 16.4 msec. and the total delay (queueing plus service) for short and long messages are 17.4 msec. and 26.4 msec. respectively.

Similar calculations are performed for the other channels and the result is summarized in Fig. 7.4.

We now calculate the length of time each transaction holds a lock, i.e. from the time the central node sends out the lock grant message to the time it receives a lock release message from the request node. Let us denote the transmission delay on channel (i,j) for long and short messages by r_{ij} and s_{ij} respectively.

Consider Class i transactions, the length of time they hold write locks, denoted WL_i, is the sum of (see Fig.7.3):

(1) transmission delay of write lock grant from central node to node i,

(2) delay due to node i sending pre-commits to copies and waiting for acknowledgement from copies, and

(3) transmission delay of lock release from node i to central node.

The length of time they hold read locks, denoted RL_i, is the sum of (see Fig. 7.3):

*For M/M/1 queues, queueing delay = $\lambda/\mu(\mu-\lambda)$ where λ is the arrival rate and μ is the service rate.

Channels C_{ij}	Additional Database Traffic short message	long message	Total Traffic short message	long msg.	Expected Queueing Delay	Expected Total Delay (Queueing plus service) short msg. $\bar{s}_{ij}$	long msg. $\bar{r}_{ij}$
C_{12}	$2\lambda_1+\lambda_2+4\lambda_4+2\lambda_5$	$\lambda_1+\lambda_2+\lambda_4+\lambda_5$	1.6	50.8	12.0 msec	13.0 msec	22.0 msec
C_{21}	$3\lambda_2$	0	.6	50.0	10.6	11.6	20.6
C_{13}	$\lambda_1+\lambda_2+\lambda_3+\lambda_5$	λ_1	.8	60.3	16.4	17.4	26.4
C_{31}	$\lambda_1+2\lambda_3+4\lambda_5+4\lambda_4$	$\lambda_1+\lambda_2+\lambda_4+\lambda_5$	1.7	50.8	26.9	28.2	39.4
C_{24}	$2\lambda_1+2\lambda_4+2\lambda_5$	$\lambda_1+\lambda_5$	1.3	30.5	40.4	42.4	60.4
C_{43}	$4\lambda_4$	$\lambda_1+\lambda_4$	.4	50.9	29.7	31.0	43.0
C_{45}	$\lambda_1+\lambda_4+\lambda_5$	0	.6	40.0	36.9	38.6	53.6
C_{35}	$\lambda_1+\lambda_3+\lambda_5$	λ_1	.6	30.3	34.5	36.5	54.5
C_{52}	0	0	0	60.0	53.3	54.6	66.3
C_{53}	$\lambda_1+3\lambda_5$	$\lambda_3+\lambda_5$	.9	55.3	42.9	44.2	56.2
C_{54}	0	λ_4	0	40.1	81.0	83.0	121.0

Figure 7.4 Additional traffic generated by DDB Management System under Centralized Locking and resultant transmission delays on the channels.

(1) transmission delay of read lock grant from central node to node i,

(2) query processing delay,

(3) transmission delay of write lock request from node i to central node,

(4) queueing delay for write locks, and

(5) WL_i

Hence, $WL_1 = s_{11} + \max.(r_{12} + r_{24} + s_{43} + s_{31},\ r_{13} + r_{35} + s_{53} + s_{31}) + s_{11}$

$$E(WL_1) = E(s_{11}) + E[\max.(r_{12} + r_{24} + s_{43} + s_{31},\ r_{13} + r_{35} + s_{53} + s_{31})] + E(s_{11})$$
$$\approx \max.[E(r_{12} + r_{24} + s_{43} + s_{31}),\ E(r_{13} + r_{35} + s_{53} + s_{31})] + 0 = 153.3 \text{ msec}$$

and, $RL_1 = s_{11} + (s_{12} + s_{24} + r_{43} + r_{31}) + s_{11} + Q(Z) + WL_1$

$$E(RL_1) = 291.1 \text{ msec} + \bar{Q}(Z)$$

where Q(I) = queueing delay for file I, I = X,Y,Z, at the central node. It is not necessary to request a lock on file Y again since the transaction already owns a lock on file Y.

Similarly, for Class 2 transactions:

$E(WL_2) = 0$ since Class 2 transactions have empty writesets

$$RL_2 = s_{12} + (s_{21} + s_{13} + r_{31} + r_{12}) + s_{21}$$

$$E(RL_2) = 115 \text{ msec.}$$

For Class 3 transactions:

$E(WL_3) = 0$

$$RL_3 = s_{13} + (s_{35} + r_{53}) + s_{31}$$

$E(RL_3) = 138.3$ msec.

For Class 4 transactions:

$$WL_4 = (s_{12} + s_{24}) + \max.(r_{43} + r_{31} + r_{12} + s_{24},\ r_{43} + r_{31} + s_{12} + s_{24}) + (s_{43} + s_{31})$$

$E(WL_4) = 261.4$ msec.

$RL_4 = (s_{12}+s_{24}) + (s_{45}+r_{54}) + (s_{43}+s_{31}) + Q(X) + WL_4$

$E(RL_4) = 535.6$ msec $+ \bar{Q}(X)$

For Class 5 transactions:

$WL_5 = (s_{13}+s_{35}) + \max.(r_{53}+s_{35}, r_{53}+r_{31}+s_{13}+s_{35},$
$r_{52}+s_{21}+s_{13}+s_{35}, r_{54}+s_{45}) + (s_{53}+s_{31})$

$E(WL_5) = 285.9$msec.

The length of time a lock is held depends on which transaction owns the lock and whether it is a write lock or a read lock. Therefore, to find the average time a lock is held, one must weight the respective lock-holding times corresponding to different transactions by their arrival rates.

Therefore, average length of time lock on file X is held = b_x
$= (\lambda_1\overline{RL_1} + \lambda_2\overline{RL_2} + \lambda_4\overline{WL_4} + \lambda_5\overline{WL_5})/(\lambda_1 + \lambda_2 + \lambda_4 + \lambda_5) = 215.2$ msec. $+ .375\bar{Q}(z)$

Similarly, average length of time lock on file Y is held = b_y
$= (\lambda_1\overline{RL_1} + \lambda_3\overline{RL_3} + \lambda_1\overline{WL_1} + \lambda_5\overline{WL_5})/(2\lambda_1 + \lambda_3 + \lambda_5) = 227.0$ msec. $+ .333\bar{Q}(Z)$

and, $b_Z = (\lambda_2\overline{RL_2} + \lambda_3\overline{RL_3} + \lambda_4\overline{RL_4} + \lambda_1\overline{WL_1} + \lambda_5\overline{WL_5})/(\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5)$
$= 215.1$ msec. $+ .111\bar{Q}(X)$

b_X, b_Y and b_Z correspond to the average service time of the queues to lock file X, Y and Z respectively. If we assume these service times to be exponential, then the service rate of the three queues are $\mu_X = 1/b_X$, $\mu_Y = 1/b_y$ and $\mu_Z = 1/b_Z$. The arrival rates of the lock requests are :

$\lambda_X = \lambda_1 + \lambda_2 + \lambda_4 + \lambda_5 = .8,$

$\lambda_Y = 2\lambda_1 + \lambda_3 + \lambda_5 = .9,$

$\lambda_Z = \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 = .9.$

$$\text{Now, } \bar{Q}(X) = \frac{\lambda_x}{\mu_x(\mu_x-\lambda_x)} = \frac{\lambda_x b_x^2}{1-\lambda_x b_x}$$

$$= \frac{\lambda_x(.2152 + .375\bar{Q}(Z))^2}{1 - \lambda_x(.2152 + .375Q(Z))} \tag{7.1}$$

$$\bar{Q}(Z) = \frac{\lambda_Z}{\mu_Z(\mu_Z-\lambda_Z)} = \frac{\lambda_Z b_Z^2}{1-\lambda_Z b_Z}$$

$$= \frac{\lambda_Z(.2151 + .111\bar{Q}(X))^2}{1-\lambda_Z(.2151 + .111\bar{Q}(X))} \tag{7.2}$$

Equ. (7.1) and (7.2) can be solved simultaneously to obtain values for $\bar{Q}(X)$ and $\bar{Q}(Z)$. An interative solution technique follows:

Initialize ; $\bar{Q}(X) = \bar{Q}(Z) = 0$

Do until ; $\bar{Q}(X)$ is close to $\bar{Q}(X)'$

$\bar{Q}(Z)$ is close to $\bar{Q}(Z)'$

Begin: $\bar{Q}(X)' = \dfrac{\lambda_X(.2152 + .375\bar{Q}(Z))^2}{1 - \lambda_X(.2152 + .375Q(Z))}$

$\bar{Q}(Z)' = \dfrac{\lambda_Z(.2151 + .111\bar{Q}(X))^2}{1 - \lambda_Z(.2151 + .111\bar{Q}(X))}$

$\bar{Q}(X) = \bar{Q}(X)'$

$\bar{Q}(Z) = \bar{Q}(Z)'$

end; (7.3)

In this case, it is found, after three iterations, that

$\bar{Q}(X) = .0547$ sec.

$\bar{Q}(Z) = .0549$ sec.

Therefore, $\mu_X = 4.24$, $\mu_Y = 4.08$, $\mu_Z = 4.52$ and the utilization of the three queues are $\rho_X = \lambda_X/\mu_X = .189$, $\rho_Y = \lambda_Y/\mu_Y = .221$, and $\rho_Z = \lambda_Z/\mu_Z = .199$ respectively.

We must next calculate the expected additional delay due to deadlocks. Suppose the deadlock detector constructs the waits-for graph every one second, i.e. on the average it takes 1/2 second to detect a deadlock.

Fig. 7.5 shows the readsets and writesets of the five Classes of transactions in our example and the potential deadlocks. To simplify the model, we are ignoring deadlocks that involve more than two transactions. In addition, as is mentioned in 6.24, Class 2 and Class 3 transactions, with empty writesets, will not enter into a deadlock with other transactions. Therefore, in our example, there are three pairs of transactions that can create deadlocks. Consider the Class 1 and Class 4 pair. Suppose T_1 arrives first. A deadlock situation is shown in Fig. 7.6(a). AB corresponds to the time between the arrival of request to lock the readset and the request to lock the writeset.

Inspection of Fig. 7.3 gives

AB = queueing delay for read locks + time read locks held - time write locks held - queueing delay for write locks

$= Q(X,Y) + RL_1 - WL_1 - Q(Z)$

where $\bar{Q}(X,Y)$ = max. queueing delay at queues X and Y

and $Q(X,Y) = \frac{\rho_X}{\mu_X(1-\rho_X)} + \frac{\rho_Y}{\mu_Y(1-\rho_Y)} - \frac{\rho_X\rho_Y}{\mu_X(1-\rho_X) + \mu_Y(1-\rho_Y)}$ (see Appendix II)

$= .118$ sec.

Hence, E(AB) = .118 sec. + .2911 sec. - .1533 sec. = .2558 sec.

The symmetric situation of a Class 4 transaction arriving first and deadlocked with a Class 1 transaction which arrives later is shown in Fig. 7.6(b).

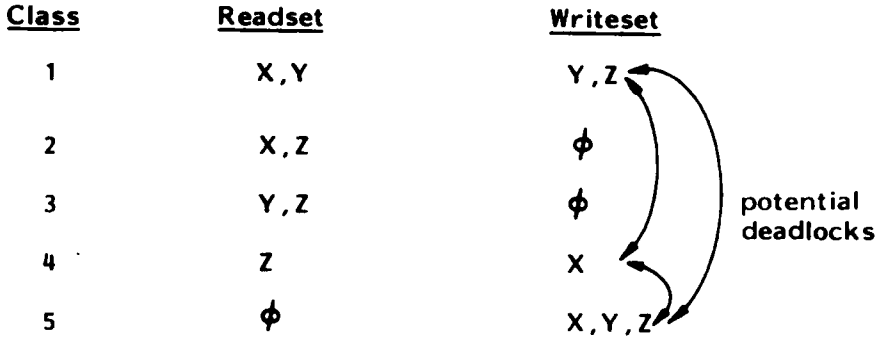
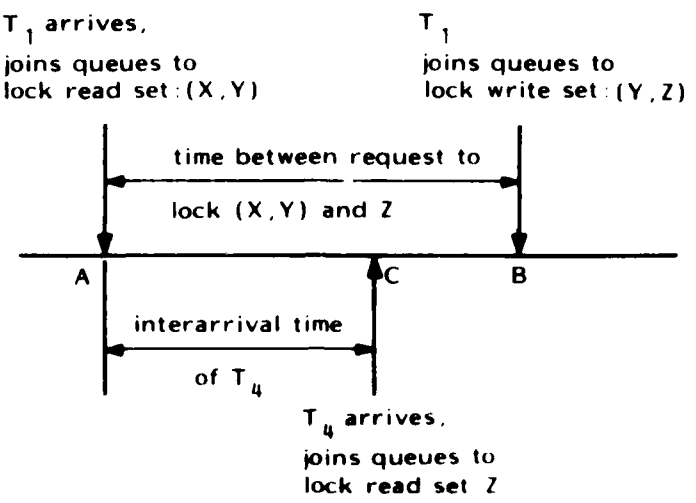
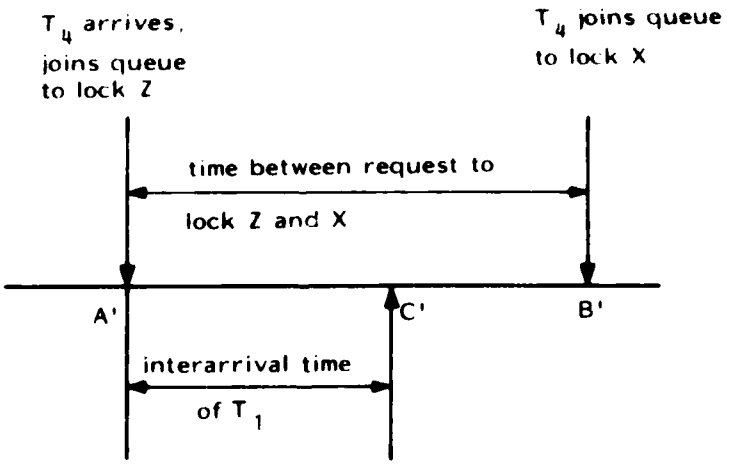


Figure 7.5 Potential Deadlocks for Example DDB



(a) T_1 arrives before T_4



(b) T_4 arrives before T_1

Figure 7.6 Finding Probability of Ie locks

$A'B' = Q(Z) + RL_4 - WL_4 - Q(X)$

$E(A'B') = .0549$ sec. $+ .5356$ sec. $- .2614$ sec. $= .3291$ sec.

Let DD_{ij} = expected delay for T_i due to possible deadlock between T_i and T_j.

From Equ. (6.10) and (6.11) respectively, we find

$$DD_{14} = \frac{1}{2}\ \frac{.3}{.3+1}\ \frac{.1}{.1+1/.2558} + (.3291+\frac{1}{2})\ \frac{.1}{.3+.1}\ \frac{.3}{.3+1/.3291} = .028 \text{ sec.}$$

and $DD_{41} = .027$ sec.

Similarly, we can calculate the expected delay due to the other two potential deadlocks.

$$DD_{15} = \frac{1}{2}\ \frac{.3}{.3+.2}\ \frac{.2}{.2+1/.2558} = .0146 \text{ sec.}$$

$$DD_{51} = (.2558+\frac{1}{2})\ \frac{.3}{.3+.2}\ \frac{.2}{.2+1/.2558} = .0221 \text{ sec.}$$

$$DD_{45} = \frac{1}{2}\ \frac{.1}{.1+.2}\ \frac{.2}{.2+1/.3291} = .0103 \text{ sec.}$$

$$DD_{54} = (.3291+\frac{1}{2})\ \frac{.1}{.1+.2}\ \frac{.2}{.2+1/.3291} = .0171 \text{ sec.}$$

(Note that there is no possibility of deadlock when the Class 5 transaction, which has empty readset, arrives first).

We can now calculate the response time for the different transaction classes. For Class i transactions, average response time under Centralized Locking, RCL_i = lock request transmission time + queueing time for locks + time locks held - lock release transmission time + expected delay due to deadlock (See Fig. 7.3)

Therefore, $RCL_1 = \bar{s}_{11} + \bar{Q}(X,Y) + \overline{RL}_1 - \bar{s}_{11} + DD_{14} + DD_{15}$

$= 0 + .118$ sec. $+ (.2911+.0549)$sec. $- 0 + .028$ sec. $+ .0146$ sec. $= .507$ sec.

$RCL_2 = \bar{s}_{21} + \bar{Q}(X,Z) + \overline{RL}_2 - \bar{s}_{21} = .0116$ sec. $+ .1045$ sec. $+ .115$sec. $-.0116$ sec. $= .220$ sec.

$$RCL_3 = \bar{s}_{31} + \bar{Q}(Y,Z) + \overline{RL}_3 - \bar{s}_{31} = .0282 \text{ sec.} + .118 \text{ sec.} + .1383 \text{ sec.}$$

$$- .0282 \text{ sec.} = .256 \text{ sec.}$$

$$RCL_4 = \bar{s}_{43} + \bar{s}_{31} + \bar{Q}(Z) + RL_4 - \bar{s}_{43} - \bar{s}_{31} + DD_{41} + DD_{45}$$

$$= .031 \text{ sec.} + .0282 \text{ sec.} + .0549 \text{ sec.} + (.5356+.0547) \text{ sec.}$$

$$- .013 \text{ sec.} - .0282 \text{ sec.} + .027 \text{ sec.} + .103 \text{ sec.} = .775 \text{ sec.}$$

$$RCL_5 = \bar{s}_{53} + \bar{s}_{31} + \bar{Q}(X,Y,Z) + \overline{RL}_5 - \bar{s}_{53} - \bar{s}_{31} + DD_{51} + DD_{54}$$

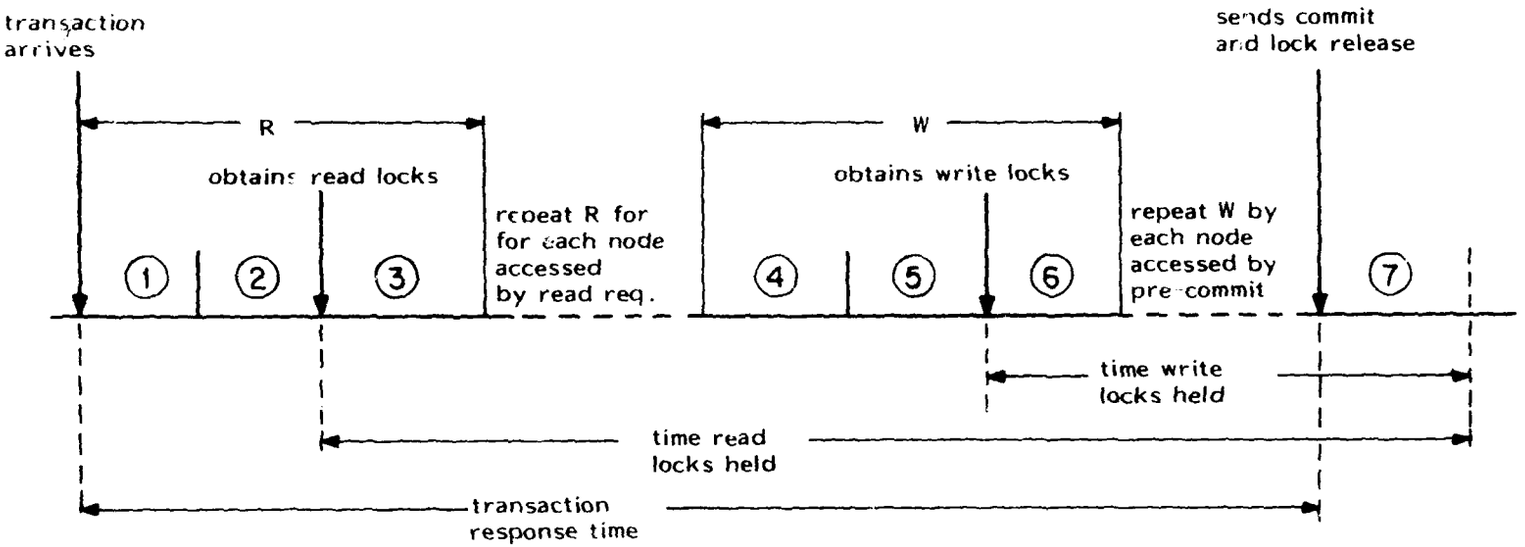
$$= .0442 \text{ sec.} + .0282 \text{ sec.} + .162 \text{ sec.} + .2859 \text{ sec}$$

$$- .0442 \text{ sec.} - .0282 \text{ sec.} + .0221 \text{ sec.} + .0171 \text{ sec.} = .506 \text{ sec.}$$

7.2 Distributed Two-Phase Locking with Ordered Queues for Deadlock Prevention

In Distributed Locking, there is no central node, and a transaction requests lock at the node where the data item is located. Compared to Centralized Locking, this algorithm is superior in that (1) there is no central node which is the bottleneck in Centralized Locking, and (2) less messages will be generated since read lock request and read request messages can be combined into one read message. (This is not possible in Centralized Locking since the central node might be different from the node where one wants to read a file copy.) The major drawback of Distributed Locking is that deadlock detection is no longer feasible.

Consider the example shown in Fig. 7.1. Suppose Ordered Queues is used to prevent deadlocks, i.e. all transactions are required to request locks in a specific order, say lock file X first, then file Y and then file Z. This means that when a transaction wants to access files at different nodes, say file X at node 1 and file Y at node 4, it must send the lock requests in serial order, i.e. request lock X first, and, after receiving the lock grant from node 1, request lock Y. If it wants to access files located at the same node, say files Y and Z at node 5, it can send the lock requests simultaneously as one message. However, at node 5, it must wait for Y first, then Z. Fig. 7.2 shows the nodes accessed by the different classes of transactions to read and write data. Let RN_i be the set of nodes that Class i transactions access to read, and WN_i be the set of nodes that Class i transactions access to write. We now consider the message flow generated by the DDB management system under Distributed Locking on behalf of the transactions. Fig. 7.7 summarizes the sequence of events corresponding to the processing of Class k transactions:



Key: ① req. node sends read req. to node i, i∈RN
② queues for read lock at node i
③ sends file from node i to req. node
④ req. node sends pre-commit to node j, j∈WN
⑤ queues for write lock at node j
⑥ node j sends acknowledgement to req. node
⑦ req. node sends lock release to nodes i and j

Figure 7.7 Chronological events corresponding to Transaction Processing under Distributed Two-phase Locking with Ordered Queues for Deadlock Prevention

(A) For each node $i \in RN_k$, repeat the following in serial order, i.e. send read messages to file X first, then Y, then Z:

(1) send read requests to node i, the read lock request is piggy-backed with this read request.

(2) queue for read locks at node i

(3) node i sends lock grant to request node and initiates file transfer

(B) For each node $j \in WN_k$, repeat the following in serial order:

(1) send pre-commits to node j. The write lock request is piggy-backed with the pre-commit.

(2) queue for write lock at node j.

(3) node j sends acknowledgement messages to request node.

(C) The request node sends commit and lock release messages to all file copies in the writeset, and lock release messages to copies read by the transaction.

The additional traffic generated by the DDB management system on behalf of the transactions can be estimated, and the transmission delay for long and short messages can be calculated. The procedure is described in detail in section 7.1 and will not be repeated here. Fig. 7.8 summarizes the results of these calculations.

We next calculate the length of time each transaction holds a lock. Let RL_{iWj} be the length of time a Class i transaction holds a read lock on file W at node j, WL_{iWj} be the time it holds a write lock, and $Q_j(W)$ be the queueing time for file W at node j, then for Class 1 transactions: (See Fig. 7.7 and Fig. 7.8)

$WL_{1Z3} = s_{31} + s_{13}$, $\overline{WL}_{1Z3} = 24.87 + 17.23 = 42.10$ msec.

Channels C_{ij}	Additional Database Traffic short message	long message	Total Traffic short message	long msg.	Expected Queueing Delay	Expected Total Delay (Queueing plus service) short msg. $\bar{s}_{ij}$	long msg. $\bar{r}_{ij}$
C_{12}	$2\lambda_1+\lambda_3+3\lambda_4$	$\lambda_1+\lambda_2+\lambda_4$	1.0	50.6	11.26msec	12.26 msec	21.26 msec
C_{21}	$\lambda_2+\lambda_5$	λ_5	.5	50.2	10.57	11.57	20.57
C_{13}	$\lambda_1+\lambda_2+\lambda_5$	λ_1	.7	60.3	16.23	17.23	26.23
C_{31}	$2\lambda_1+\lambda_3+2\lambda_4$	$\lambda_2+\lambda_4$	.9	50.3	23.62	24.87	36.11
C_{24}	$2\lambda_1+\lambda_3+\lambda_4$	λ_1	.8	30.3	35.89	37.89	55.89
C_{43}	$2\lambda_4$	$\lambda_1+\lambda_3+\lambda_4$	.2	51.0	29.11	30.44	42.44
C_{45}	$\lambda_1+\lambda_4+\lambda_5$	λ_1	.6	40.3	37 73	39.40	54.40
C_{35}	$\lambda_1+2\lambda_5$	0	.7	30.0	34.30	36.30	54.30
C_{52}	0	λ_5	0	60.2	54.23	55.57	67.57
C_{53}	λ_1	λ_5	.3	55.2	38.86	40.19	52.19
C_{54}	0	$\lambda_4+\lambda_5$	0	40.3	83.09	85.09	103.09

Figure 7.8 Additional traffic generated by DDB management system under Distributed Locking with Ordered Queues for Deadlock Detection and resultant transmission delays on the channels.

$$WL_{1Y5} = WL_{1Z5} = (s_{52} + s_{21}) + r_{13} + Q_3(Z) + WL_{1Z3} - s_{13} + (s_{13} + s_{35})$$

$$\overline{WL}_{1Y5} = \overline{WL}_{1Z5} = 171.77 \text{ msec.} + \overline{Q}_3(Z)$$

$$WL_{1Y4} = (s_{43} + s_{31}) + (r_{13} + r_{35}) + Q_5(Y,Z) + WL\ 1Y5 - (s_{13} + s_{35}) + (s_{43} + s_{31})$$

$$\overline{WL}_{1Y4} = 309.39 \text{ msec.} + \overline{Q}_3(Z) + \overline{Q}_5(Y,Z)$$

$$RL_{1Y4} = (r_{43} + r_{31}) + WL_{1Y4}$$

$$\overline{RL}_{1Y4} = 387.94 \text{ msec.} + \overline{Q}_3(Z) + \overline{Q}_5(Y,Z)$$

$$RL_{1X1} = r_{11} + (s_{12} + s_{24}) + Q_4(Y) + RL_{1Y4}$$

$$\overline{RL}_{1X1} = 438.09 \text{ msec.} + \overline{Q}_4(Y) + \overline{Q}_3(Z) + \overline{Q}_5(Y,Z)$$

Similarly, for Class 2 transactions,

$$RL_{2Z3} = (r_{31} + r_{12}),\ \overline{RL}_{2Z3} = 57.37 \text{ msec.}$$

$$RL_{2X2} = r_{22} + Q_3(Z) + RL_{2Z3}$$

$$\overline{RL}_{2X2} = 57.37 \text{ msec.} + \overline{Q}_3(Z)$$

For Class 3 transactions,

$$RL_{3Z3} = r_{33} = 0$$

$$RL_{3Y4} = r_{43} + Q_3(Z) + RL_{3Z3}$$

$$\overline{RL}_{3Y4} = 42.44 \text{ msec.} + \overline{Q}_3(Z)$$

Class 4 transactions are slightly different. Since the writeset contains file X and the readset contains file Z, to maintain the serial order of locking file X first, then Y and then Z, it is necessary to send lock X messages before sending read Z requests. (Recall that we normally send write lock requests piggy-backed with pre-commit messages). Therefore, for Class A transactions only, the sequence of events becomes:

(1) send write lock requests to all copies of file X

(2) send read request to node 5 to read Z

(3) send pre-commits (without lock requests) to all copies of file X

(4) copies send acknowledgement to node 4

(5) send commits and lock releases

Hence, $RL_{4Z5} = r_{54} + (r_{43} + r_{31} + r_{12}) + (s_{12} + s_{24}) + (s_{43} + s_{31} + s_{12})$

$\overline{RL}_{4Z5} = 320.62$ msec.

$WL_{4X1} = (s_{12} + s_{24}) + s_{45} + Q_5(Z) + RL_{4Z5}$

$\overline{WL}_{4X1} = 410.17$ msec. $+ \overline{Q}_5(Z)$

$WL_{4X2} = s_{24} + s_{24} + s_{45} + Q_5(Z) + RL_{4Z5}$

$\overline{WL}_{4X2} = 435.80$ msec. $+ \overline{Q}_5(Z)$

$WL_{4X4} = s_{44} + s_{44} + Q_5(Z) + RL_{4Z5}$

$\overline{WL}_{4X4} = 320.62$ msec. $+ \overline{Q}_5(Z)$

For Class 5 transactions,

$WL_{5Z3} = s_{35} + s_{53}$

$\overline{WL}_{5Z3} = 76.49$ msec.

$WL_{5Z5} = WL_{5Y5} = s_{55} + r_{53} + Q_3(Z) + s_{35} + s_{55}$

$\overline{WL}_{5Z5} = 88.49$ msec. $+ \overline{Q}_3(Z)$

$WL_{5X4} = WL_{5Y4} = s_{45} + r_{55} + Q_5(Y,Z) + WL_{5Z5} - s_{55} + s_{45}$

$\overline{WL}_{5X4} = \overline{WL}_{5Y4} = 167.29$ msec. $+ \overline{Q}_3(Z) + \overline{Q}_5(Y,Z)$

$WL_{5X1} = (s_{13} + s_{35}) + r_{54} + Q_4(X,Y) + WL_{5X4} - s_{45} + (s_{13} + s_{35})$

$$\overline{WL}_{5X1} = 338.04 \text{ msec.} + \bar{Q}_3(Z) + \bar{Q}_5(Y,Z) + \bar{Q}_4(X,Y)$$

$$WL_{5X2} = (s_{21} + s_{13} + s_{35}) + r_{54} + Q_4(X,Y) + WL_{5X4} - s_{45} + (s_{21} + s_{13} + s_{35})$$

$$\overline{WL}_{5X2} = 361.18 \text{ msec.} + \bar{Q}_3(Z) + \bar{Q}_5(Y,Z) + \bar{Q}_4(X,Y)$$

The length of time a lock is held depends on which transaction owns the lock. Therefore, to find the average length of time a lock is held at each node, we must weight the respective lock-holding times corresponding to different transactions by their arrival rates.

Let b_{Wj} be the average length of time lock on file W is held at node j. This corresponds to the in-service time of a transaction holding a lock on file W at node j, without accounting for delay due to blocking. (See section 6.2.1)

$$b_{X1} = (\lambda_1\overline{RL}_{1X1} + \lambda_4\overline{WL}_{4X1} + \lambda_5\overline{WL}_{5X1}) \; / \; (\lambda_1 + \lambda_4 + \lambda_5)$$

$$= 400.09 \text{ msec.} + .5\bar{Q}_4(Y) + .833\bar{Q}_3(Z) + .1667\bar{Q}_5(Z) + .333\bar{Q}_4(X,Y) + .833\bar{Q}_5(Y,Z)$$

$$b_{X2} = (\lambda_2 RL_{2X2} + \lambda_4\overline{WL}_{4X2} + \lambda_5\overline{WL}_{5X2}) \; / \; (\lambda_2 + \lambda_4 + \lambda_5)$$

$$= 254.58 \text{ msec.} + .8\bar{Q}_3(Z) + .2\bar{Q}_5(Z) + .4\bar{Q}_5(Y,Z) + .4\bar{Q}_4(X,Y)$$

$$b_{Z3} = (\lambda_1\overline{WL}_{1Z3} + \lambda_2\overline{RL}_{2Z3} + \lambda_3\overline{RL}_{3Z3} + \lambda_5\overline{WL}_{5Z3}) \; / \; (\lambda_1 + \lambda_2 + \lambda_3 + \lambda_5)$$

$$= 49.26 \text{ msec.}$$

$$b_{X4} = (\lambda_4\overline{WL}_{4X4} + \lambda_5\overline{WL}_{5X4}) \; / \; (\lambda_4 + \lambda_5)$$

$$= 218.40 \text{ msec.} + .1\bar{Q}_5(Z) + .2\bar{Q}_3(Z) + .2\bar{Q}_5(Y,Z)$$

$$b_{Y4} = (\lambda_1\overline{RL}_{1Y4} + \lambda_1\overline{WL}_{1Y4} + \lambda_3\overline{RL}_{3Y4} + \lambda_5\overline{WL}_{5Y4}) \; / \; (2\lambda_1 + \lambda_3 + \lambda_5)$$

$$= 506.78 \text{ msec.} + \bar{Q}_3(Z) + .8889\bar{Q}_5(Y,Z)$$

$$b_{Y5} = (\lambda_1 \overline{WL}_{1Y5} + \lambda_5 \overline{WL}_{5Y5}) \; / \; (\lambda_1 + \lambda_5)$$

$$= 138.46 \text{ msec.} + \bar{Q}_3(Z)$$

$$b_{Z5} = (\lambda_1 \overline{WL}_{1Z5} + \lambda_4 \overline{RL}_{4Z5} + \lambda_5 \overline{WL}_{5Z5}) \; / \; (\lambda_1 + \lambda_4 + \lambda_5)$$

$$= 168.82 \text{ msec.} + .8333\bar{Q}_3(Z)$$

In our example, it is noted that nodes 1, 2 and 3 contains only one file each. Therefore, the in-service times at these nodes are given by b_{X1}, b_{X2} and b_{Z3}. However, at nodes 4 and 5, there are two files each. When a transaction accesses these files, the locks must be requested in serial order. Thus the in-service time of locking file W at node j when the serial locking order is observed, denoted by a_{Wj}, must be calculated as described in 6.2.2. For example, at node 5, $a_{Z5} = b_{Z5}$, but $a_{Y5} = b_{Y5} + S_{Y5}$, i.e. the in-service time at the queue for file Y plus the total service time at the queue for file Z. The queueing network (described in 6.2.2) corresponding to node 5 is shown in Fig. 7.9(a).

Let λ_{Mj} and μ_{Mj} be the arrival and service rates of lock requests for file M at node j, w_{Mj} be the average queueing time, S_{Mj} be the average total service time in the queue to lock file M at node j. Consider node 3,

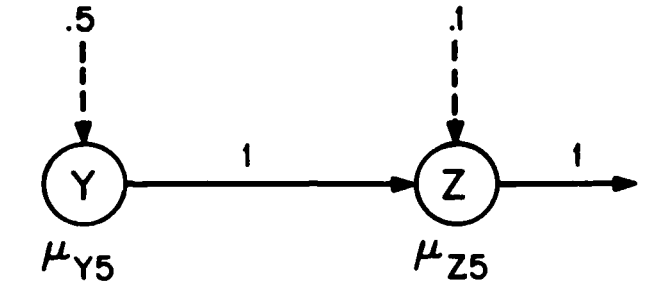
$$\mu_{Z3} = 1/b_{Z3} = 20.30$$

$$\bar{Q}_3(Z) = \frac{\lambda_{Z3}}{\mu_{Z3}(\mu_{Z3} - \lambda_{Z3})} = \frac{.8}{20.30(20.30 - .8)} = .00202 \text{ sec.}$$

Consider node 5, $b_{Y5} = 138.46 \text{ msec.} + \bar{Q}_3(Z)$

$$= .14048 \text{ sec.}$$

$$a_{Z5} = b_{Z5} = 168.82 \text{ msec.} + .8333\bar{Q}_3(Z) = .1705 \text{ sec.}$$



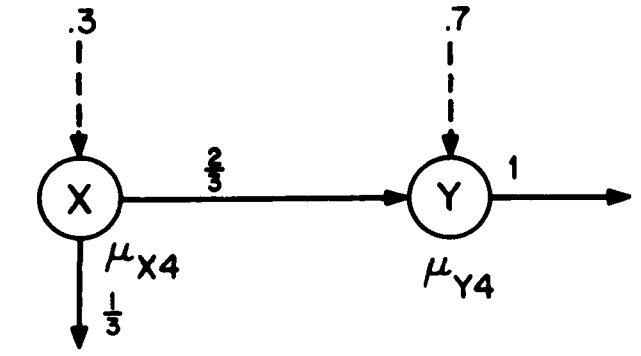
$\lambda_{Y5} = .5$

$\lambda_{Z5} = .5 + .1 = .6$

EXTERNAL ARRIVALS

ROUTING PROBABILITIES

(a) Queueing network at node 5



$\lambda_{X4} = .3$

$\lambda_{Y4} = .3(\frac{2}{3}) + .7 = .9$

EXTERNAL ARRIVALS

ROUTING PROBABILITIES

(b) Queueing network at node 4

Figure 7.9 Queueing Network Models for Nodes 4 and 5 in Example DDB

$$S_{Z5} = \frac{1}{\mu_{Z5} - \lambda_{Z5}} = \frac{1}{1/.1705 - .6} = .1899 \text{ sec.}$$

$$\bar{Q}_5(Z) = w_{Z5} = S_{Z5} - a_{Z5} = (.1899 - .1705) \text{ sec.} = .0194 \text{ sec.}$$

$$a_{Y5} = b_{Y5} + S_{Z5} = .14048 \text{ sec.} + .1899 \text{ sec.} = .33038 \text{ sec.}$$

Assuming that the service time is still exponential,

$$\mu_{Y5} = 1/a_{Y5} = 2.996, \text{ and}$$

$$\bar{Q}_5(Y) = \frac{\lambda_{Y5}}{\mu_{Y5}(\mu_{Y5} - \lambda_{Y5})} = \frac{.5}{2.996(2.996 - .5)} = .06537 \text{ sec.}$$

$\bar{Q}_5(Y,Z)$, the queueing time for both files Y and Z,

$$= \bar{Q}_5(Y) + \bar{Q}_5(Z) = .08477 \text{ sec.}$$ since requests for files Y and Z must be queued serially.

Consider node 4, $b_{X4} = 218.40 \text{ msec.} + .1\bar{Q}_5(Z) + .2\bar{Q}_3(Z) + .2\bar{Q}_5(Y,Z)$

$$= .2377 \text{ sec.}$$

$$a_{Y4} = b_{Y4} = 506.78 \text{ msec.} + \bar{Q}_3(Z) + .8889\bar{Q}_5(Y,Z)$$

$$= .5842 \text{ sec.}$$

$$S_{Y4} = \frac{1}{\mu_{Y4} - \lambda_{Y4}} = \frac{1}{1/.5842 - .9} = 1.2317 \text{ sec.}$$

$$\bar{Q}_4(Y) = w_{Y4} = S_{Y4} - a_{Y4} = .6475 \text{ sec.}$$

$$a_{X4} = \frac{1}{3} b_{X4} + \frac{2}{3} (b_{X4} + S_{Y4}) = 1.0588 \text{ sec.}$$

Assuming that the service time is still exponential,

$$\mu_{X4} = 1/a_{X4} = .94447, \text{ and}$$

$$\bar{Q}_4(X) = \frac{\lambda_{X4}}{\mu_{X4}(\mu_{X4} - \lambda_{X4})} = \frac{.3}{.94447(.94447 - .3)} = .4929 \text{ sec.}$$

$$\bar{Q}_4(X,Y) = \bar{Q}_4(X) + \bar{Q}_4(Y) = 1.140 \text{ sec.}$$

Consider node 1, $b_{X1} = 1.170$ sec.

Therefore, $\bar{Q}_1(X) = \frac{\lambda_{X1}}{\mu_{X1}(\mu_{X1} - \lambda_{X1})}$ = 2.85 sec.

Consider node 2, b_{X2} = .7500 sec.

Therefore, $\bar{Q}_2(X) = \frac{\lambda_{X2}}{\mu_{X2}(\mu_{X2} - \lambda_{X2})}$ = .4500 sec.

We can now calculate the response time of the different transaction classes. For Class i transactions, average response time under Distributed Locking with Ordered Queues for Deadlock Prevention, $RDLOQ_i$ = read request transmission time + queueing time for locks at first node accessed + time locks held - lock release transmission time = queueing time for locks at first node accessed + time locks held. (See Fig. 7.7)

Therefore, $RDLOQ_1 = \bar{Q}_1(X) + RL_{1X1}$

= 2.85 sec. + 1.172 sec. = 4.02 sec.

$RDLOQ_2 = \bar{Q}_2(X) + RL_{2X2}$

= .45 sec. + .05939 sec. = .509 sec.

$RDLOQ_3 = \bar{Q}_4(Y) + RL_{3Y4}$

= .6475 sec. + .04446 sec. = .692 sec.

$RDLOQ_4 = E[\max(Q_1(X) + WL_{4X1}, Q_2(X) + WL_{4X2}, Q_4(X) + WL_{4X4})]$

$\simeq \bar{Q}_1(X) + \overline{WL}_{4X1}$ = 2.85 sec. + .4296 sec. = 3.28 sec.

$RDLOQ_5 = E[\max(Q_1(X) + WL_{5X1}, Q_2(X) + WL_{5X2}, Q_4(X,Y) + WL_{5X4})]$

$\simeq \bar{Q}_1(X) + \overline{WL}_{5X1}$ = 2.85 sec. + 1.56 sec. = 4.41 sec.

7.3 Distributed Two-Phase Lock with Prioritized Transactions for Deadlock Prevention

In this example, Prioritized Transactions will be used to prevent deadlocks. This is more efficient than the Ordered Queues scheme in that more concurrency is possible. Whereas in the Ordered Queues scheme, locks have to be obtained in serial order, one after another, in the Prioritized Transactions scheme they can be obtained simultaneously. The tradeoff, however, is that for Prioritized Transactions, it is sometimes necessary to restart some transactions.

Fig. 7.10 summarizes the sequence of events corresponding to the processing of Class k transactions:

(A) For each node $i \varepsilon RN_k$ messages, the following is repeated:

(1) send read request to node i, lock request is piggy-backed with this read request.

(2) queue for read lock at node i

(3) node sends lock grant to request node and initiates file transfer.

Note that the nodes will be accessed simultaneously and the delay associated with query processing is $\max_{i \varepsilon RN_k} R_i$ (See Fig. 7.10), where R_i is the delay associated with accessing node i.

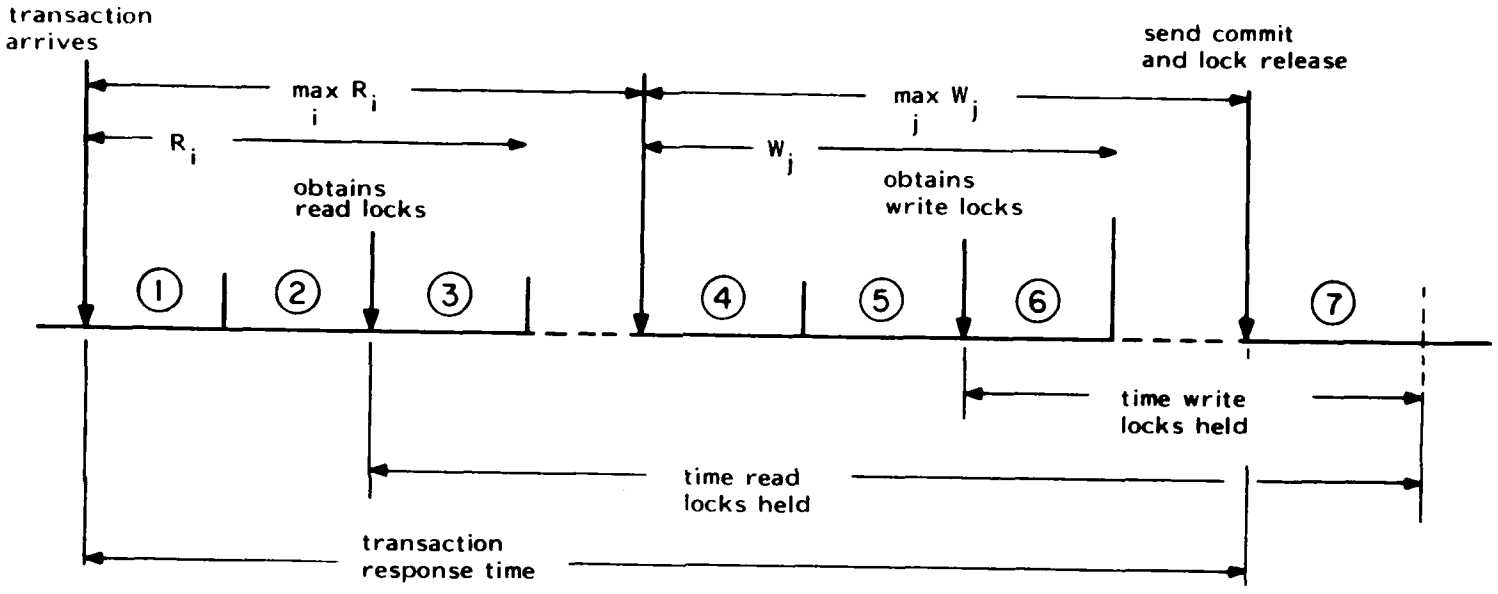
(B) For each node $j \varepsilon WN_k$, the following is repeated:

(1) send pre-commit to node j. The write lock request is piggy-backed with it.

(2) queue for write lock at node j

(3) node j sends acknowledgement to request node.

Again, all copies will be accessed simultaneously and the delay associated with pre-commit is $\max_{j \cdot WN_k} W_j$ (See Fig.7.10), where W_j is the delay associated with accessing node j.



Key: ① sends read request to node i from req. node, $i \in RN$

② queues for read lock at node i

③ sends file from node i to req. node

④ sends pre commit to node j from req. node, $j \in WN$

⑤ queues for write lock at node j

⑥ node j sends acknowledgement to req. node

⑦ req. node sends lock release to nodes i and j

Figure 7.10 Chronological Events corresponding to Transaction Processing under Distributed Two-phase Locking with Prioritized Transactions for Deadlock Prevention

(C) The request node sends commit and lock release messages.

The volume of messages generated in the communication subnetwork is similar to that described in section 7.2. It is therefore assumed that the average delays on the communication channels are the same. (See Fig.7.8)

We next calculate the length of time each transaction holds a lock. Let RL_{iWj} be the length of time a class i transaction holds a read lock on file W at node j, WL_{iWj} be the time it holds a write lock, $Q_j(W)$ be the queueing time for file W at node j, $MW_i = \max_{j \epsilon WN_i} W_j$ be the delay associated with pre-commit for Class i transactions, and $MR_i = \max_{j \epsilon RN_i} R_j$ be the delay associated with query processing for Class i transactions, then for Class 1 transactions: (see Fig.7.10)

WL_{123} = delay due to pre-commit + lock release transmission time - pre-commit - queueing time for write lock

$MW_1 + s_{13} + s_{35} - r_{13} - r_{35} - Q_5(Z)$

$\bar{MW} - \bar{Q} - 27$ msec.

$MW_1 + s_{13} + s_{35} - r_{13} - r_{35} - Q_5(Y)$

$\bar{MW}_1 - \bar{Q}_5(Y) - 27$ msec.

$MW_1 + s_{13} + s_{35} - r_{13} - r_{35} - Q_5(Z)$

$\bar{MW}_1 - \bar{Q}_5(Z) - 27$ msec.

$MW_1 + s_{12} + s_{24} - r_{12} - r_{24} - Q_4(Y)$

$\bar{MW}_1 - \bar{Q}_4(Y) - 27$ msec.

RL_{134} = delay due to query processing + delay due to pre-commit + lock release transmission time - read request transmission time - queueing delay

$MR_1 + s_{13} + s_{34} - r_{13} - r_{34} - Q_4(Y)$

Therefore, $RL_{1Y4} = MR_1 + MW_1 + (s_{12} + s_{24}) - (s_{12} + s_{24}) - Q_4(Y)$

$\overline{RL}_{1Y4} = \overline{MR}_1 + \overline{MW}_1 - \overline{Q}_4(Y)$

$\overline{RL}_{1X1} = \overline{MR}_1 + \overline{MW}_1 - \overline{Q}_1(X)$

For Class 2 transactions:

$RL_{2Z3} = MR_2 + (s_{21} + s_{13}) - (s_{21} + s_{13}) - Q_3(Z)$

$\overline{RL}_{2Z3} = \overline{MR}_2 - \overline{Q}_3(Z)$

$\overline{RL}_{2X2} = \overline{MR}_2 - \overline{Q}_2(X)$

For Class 3 transactions:

$\overline{RL}_{3Z3} = \overline{MR}_3 - \overline{Q}_3(Z)$

$RL_{3Y4} = MR_3 + (s_{35} + s_{54}) - (s_{35} + s_{54}) - Q_4(Y)$

$\overline{RL}_{3Y4} = \overline{MR}_3 - \overline{Q}_4(Y)$

For Class 4 transactions:

$\overline{WL}_{4X4} = \overline{MW}_4 - \overline{Q}_4(X)$

$WL_{4X1} = MW_4 + (s_{43} + s_{31}) - (r_{43} - r_{31}) - Q_1(X)$

$\overline{WL}_{4X1} = \overline{MW}_4 - \overline{Q}_1(X)$ -23.24 msec.

$WL_{4X2} = MW_4 + (s_{45} + s_{52}) - (r_{45} - r_{52}) - Q_2(X)$

$\overline{WL}_{4X2} = \overline{MW}_4 - \overline{Q}_2(X)$ - 27 msec.

$RL_{4Z5} = MR_4 + MW_4 + s_{45} - s_{45} - Q_5(Z)$

$\overline{RL}_{4Z5} = \overline{MR}_4 + \overline{MW}_4 - \overline{Q}_5(Z)$

For Class 5 transactions:

$WL_{5Z3} = MW_5 + s_{53} - r_{53} - Q_3(Z)$

$$\overline{WL}_{5Z3} = \overline{MW}_5 - \overline{Q}_3(Z) - 12 \text{ msec.}$$

$$\overline{WL}_{5Z5} = \overline{MW}_5 - \overline{Q}_5(Z)$$

$$\overline{WL}_{5Y5} = \overline{MW}_5 - \overline{Q}_5(Y)$$

$$WL_{5X4} = MW_5 + s_{54} - r_{54} - Q_4(X)$$

$$\overline{WL}_{5X4} = \overline{MW}_5 - \overline{Q}_4(X) - 18 \text{ msec.}$$

$$WL_{5Y4} = MW_5 + s_{54} - r_{54} - Q_4(Y)$$

$$\overline{WL}_{5Y4} = \overline{MW}_5 - \overline{Q}_4(Y) - 18 \text{ msec.}$$

$$WL_{5X1} = MW_5 + (s_{53} + s_{31}) + (r_{53} + r_{31}) - Q_1(X)$$

$$\overline{WL}_{5X1} = \overline{MW}_5 - \overline{Q}_1(X) - 23.24 \text{ msec}$$

$$WL_{5X2} = MW_5 + s_{52} - r_{52} - Q_2(X)$$

$$\overline{WL}_{5X2} = \overline{MW}_5 - \overline{Q}_2(X) - 12 \text{ msec.}$$

We next find the length of time a lock is held at the different nodes. Let b_{Wj} be the average length of time lock on file W is held at node j.

$$b_{X1} = (\lambda_1\overline{RL}_{1X1} + \lambda_4\overline{WL}_{4X1} + \lambda_5\overline{WL}_{5X1}) \; / \; (\lambda_1 + \lambda_4 + \lambda_5)$$

$$= .5\overline{MR}_1 + .5\overline{MW}_1 + .1667\overline{MW}_4 + .333\overline{MW}_5 - \overline{Q}_1(X) - 11.62 \text{ msec.}$$

$$b_{X2} = (\lambda_2\overline{RL}_{2X2} + \lambda_4\overline{WL}_{4X2} + \lambda_5\overline{WL}_{5X2}) \; / \; (\lambda_2 + \lambda_4 + \lambda_5)$$

$$= .4\overline{MR}_2 + .2\overline{MW}_4 + .4\overline{MW}_5 - \overline{Q}_2(X) - 10.2 \text{ msec.}$$

$$b_{Z3} = (\lambda_1\overline{WL}_{1Z3} + \lambda_2\overline{RL}_{2Z3} + \lambda_3\overline{RL}_{3Z3} + \lambda_5\overline{WL}_{5Z3}) \; / \; (\lambda_1 + \lambda_2 + \lambda_3 + \lambda_5)$$

$$= .375\overline{MW}_1 + .25\overline{MR}_2 + .125\overline{MR}_3 + .25\overline{MW}_5 - \overline{Q}_3(Z) - 6.375 \text{ msec.}$$

$$b_{X4} = (\lambda_4\overline{WL}_{4X4} + \lambda_5\overline{WL}_{5X4}) \; / \; (\lambda_4 + \lambda_5)$$

$$= .333\overline{MW}_4 + .667\overline{MW}_5 - \overline{Q}_4(X) - 12 \text{ msec.}$$

$$b_{Y4} = (\lambda_1\overline{RL}_{1Y4} + \lambda_1\overline{WL}_{1Y4} + \lambda_3\overline{RL}_{3Y4} + \lambda_5\overline{WL}_{5Y4}) \; / \; (2\lambda_1 + \lambda_3 + \lambda_5)$$

$$= .333\overline{MR}_1 + .667\overline{MW}_1 + .111\overline{MR}_3 + .222\overline{MW}_5 - \overline{Q}_4(Y) - 13 \text{ msec.}$$

$$b_{Y5} = (\lambda_1\overline{WL}_{1Y5} + \lambda_5\overline{WL}_{5Y5}) \; / \; (\lambda_1 + \lambda_5)$$

$$= .6\overline{MW}_1 + .4\overline{MW}_5 - \overline{Q}_5(Y) - 16.2 \text{ msec.}$$

$$b_{Z5} = (\lambda_1\overline{WL}_{1Z5} + \lambda_4\overline{RL}_{4Z5} + \lambda_5\overline{WL}_{5Z3}) \; / \; (\lambda_1 + \lambda_4 + \lambda_5)$$

$$= .5\overline{MW}_1 + .1667\overline{MR}_4 + .1667\overline{MW}_4 + .333\overline{MW}_5 - \overline{Q}_5(Z) - 13.5 \text{ msec.}$$

Therefore, we have seven equations in seven unknowns, namely $\overline{Q}_1(X)$, $\overline{Q}_2(X)$, $\overline{Q}_3(Z)$, $\overline{Q}_4(X)$, $\overline{Q}_4(Y)$, $\overline{Q}_5(Y)$ and $\overline{Q}_5(Z)$. (Note that b_{Wi} is related to $\overline{Q}_i(W)$ by the equation

$$\overline{Q}_i(W) = \frac{\lambda}{\mu(\mu - \lambda)} = \frac{\lambda_{Wi} b_{Wi}^2}{1 - \lambda_{Wi} b_{Wi}}$$ where λ_{Wi} is the lock request rate for file W at node i.) Of course, we have to determine $\overline{MR}_i$, $\overline{MW}_i$, i = 1,...,5 first. For example, $MR_1 = \max (Q_1(X), s_{12} + s_{24} + Q_4(Y) + r_{43} + r_{31})$

$$\overline{MR}_1 = \max (\overline{Q}_1(X), \overline{Q}_4(Y) + 128.7 \text{ msec.})$$

Expressions for $\overline{MR}_1$, $\overline{MR}_2$, etc. are difficult to obtain in closed form. Therefore, we make the additional assumption that the delay corresponding to accessing each node, i.e. the transmission time plus queueing time for locks, is exponentially distributed. Since the expected value of the maximum of several exponentials have been derived in Appendix IV, we can find closed form expressions for $\overline{MR}_1$, $\overline{MR}_2$, etc. For example, $\overline{MR}_1 = \overline{Q}_1(X) + \overline{Q}_4(Y) + 1/(1/\overline{Q}_1(X) + 1/\overline{Q}_4(Y))$.

After substituting these expressions of MR_i, MW_i into Equations (7.4), we can solve them simultaneously, using an iterative procedure.

The procedure converges after six iterations and outputs the following:

$b_{X1} = .421$ sec. $\quad Q_1(X) = .134$ sec.

$b_{X2} = .173$ sec. $\quad Q_2(X) = .164$ sec.

$b_{Z3} = .226$ sec. $\quad Q_3(Z) = .0579$ sec.

$b_{X4} = .194$ sec. $\quad Q_4(X) = .0119$ sec.

$b_{Y4} = .355$ sec. $\quad Q_4(Y) = .166$ sec.

$b_{Y5} = .256$ sec. $\quad Q_5(Y) = .0375$ sec.

$b_{Z5} = .278$ sec. $\quad Q_5(Z) = .0557$ sec.

$\overline{MR}_1 = .511$ sec. $\quad \overline{MR}_2 = .171$ sec. $\quad \overline{MR}_3 = .423$ sec.

$\overline{MR}_4 = .196$ sec. $\quad \overline{MW}_1 = .407$ sec. $\quad \overline{MW}_4 = .334$ sec.

$\overline{MW}_5 = .308$ sec.

$\overline{RL}_{1X1} = .784$ sec. $\quad \overline{RL}_{1Y4} = .752$ sec. $\quad \overline{WL}_{1Y4} = .214$ sec.

$\overline{WL}_{1Y5} = .343$ sec. $\quad \overline{WL}_{1Z3} = .340$ sec. $\quad \overline{WL}_{1Z5} = .324$ sec.

$\overline{RL}_{2X2} = .117$ sec. $\quad \overline{RL}_{2Z3} = .113$ sec. $\quad \overline{RL}_{3Y4} = .365$ sec.

$\overline{RL}_{3Z3} = .365$ sec. $\quad \overline{RL}_{4Z5} = .474$ sec. $\quad \overline{WL}_{4X1} = .198$ sec.

$\overline{WL}_{4X2} = .1[illegible]$ sec. $\quad \overline{WL}_{4X4} = .322$ sec. $\quad \overline{WL}_{5X1} = .172$ sec.

$\overline{WL}_{5X2} = .132$ sec. $\quad \overline{WL}_{5X4} = .278$ sec. $\quad \overline{WL}_{5Y4} = .124$ sec.

$\overline{WL}_{5Y5} = .271$ sec. $\quad \overline{WL}_{5Z3} = .249$ sec. $\quad \overline{WL}_{5Z5} = .252$ sec.

We next calculate the probability a transaction will be restarted under the wound-wait deadlock prevention scheme. It can be shown that for this particular example, wound-wait induces fewer restarts than wait-die. Since two transactions conflict if at least one of them is a write transaction, there are three distinct cases to consider: (1) a Class i transaction T_i owns a read lock on a data item X at node r and a Class j transaction T_j tries to get a write lock on X at the same node, (2) T_i owns a write lock on Y at node w and a Class k transaction T_k tries to get a read lock on Y at node w, and (3) T_i owns a write lock on Y at node w and a Class m transaction T_m tries to get a write lock on Y at node w.

Recall that under wound-wait, every transaction is given a timestamp (its priority) when it enters the system, and a transaction will be restarted if a conflicting transaction with higher priority (older timestamp) is forced to wait for it to release a lock.

Inspection of Fig. 7.10 lets us construct the three scenarios corresponding to transaction restarts. (See Fig. 7.11 (a), (b), (c).) Let a_i be the interarrival time of T_i. In Case 1 (Fig. 7.11(a)), T_i will be restarted if a conflicting T_j with older timestamp arrives at node r after T_i does. Therefore, PR_{ijr} = P(T_i restarted by T_j at node r)

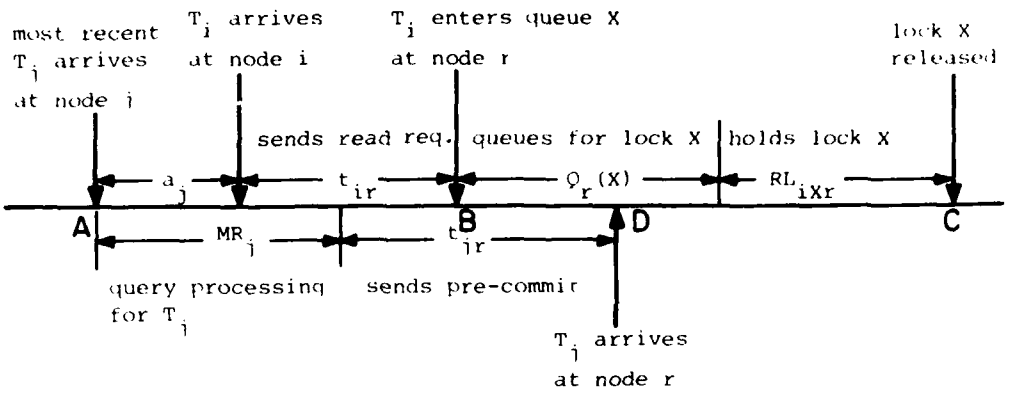
$$= P(AB < AD < AC)$$

$$= P(AB < AD)P(AD < AC \mid AB < AD)$$

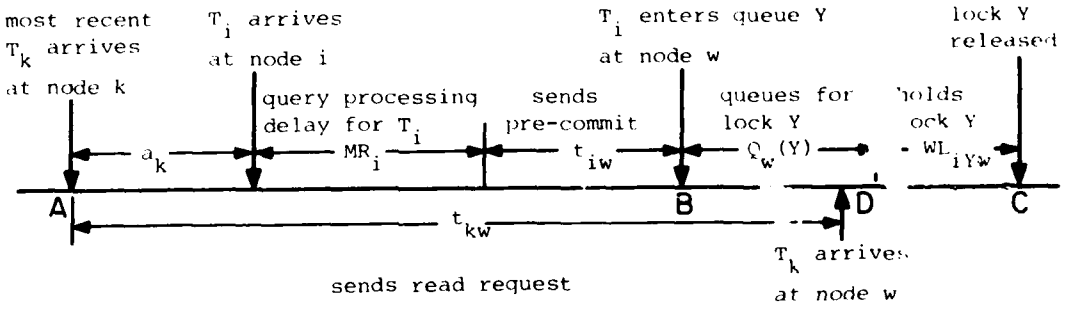
$$= P(AB < AD)P(AD - AB < AC - AB \mid AB < AD)$$

$$= P(AB < AD)P(AD - AB < BC \mid AD - AB > 0)$$

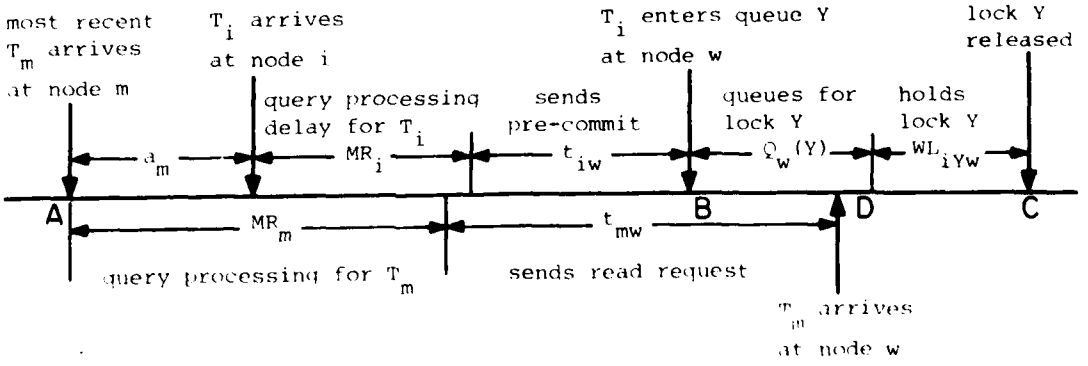
We now make the additional assumptions that AD and BC are exponentially distributed, then



(a) Case 1 : T_i restarted because of read-write conflict



(b) Case 2 : T_i restarted because of write-read conflict



(c) Case 3 : T_i restarted because of write-write conflict

Figure 7.11 Finding Probability and Delay of Transaction Restarts

$PR_{ijr} = P(MR_j + r_{jr} > a_j) \; P(MR_j + r_{jr} > s_{ir}) \cdot$

$$P(Q_r(X) + RL_{iXr} > MR_j + r_{jr}) \qquad (7.5)$$

W_{ijr} = E(delay for T_i due to rejection by T_j at node r)

= time wasted in processing the aborted transaction + transmission delay from node r to node i to report the abortion

$= E(s_{ir}) + E(BD \mid AB < AD < AC) + E(s_{ri})$

$= E(s_{ir}) + E(s_{ri}) + E(BD \mid BD < BC)$

$= E(s_{ir}) + E(s_{ri}) + E(\min.(BD, BC))$

See Appendix I for a derivation of $E(BD \mid BD < BC) = E(\min.(BD, BC))$.

Hence, $W_{ijr} = E(s_{ir}) + E(s_{ri}) + E(\min.(r_{jr} + MR_j, Q_r(X) + RL_{ixr}))$ (7.6)

In Case 2 (Fig. 7.11(b)), PR_{ikw} = P(T_i rejected by T_k at node w)

$= P(AB < AD < AC) = P(s_{kw} > a_k) \; P(s_{kw} > MR_i) \; P(s_{kw} > r_{iw}) \cdot$

$$P(Q_w(Y) + WL_{iyw} > s_{kw}) \qquad (7.7)$$

and W_{ikw} = E(delay for T_i due to rejection by T_k at node w)

$$= E(s_{wi}) + E(MR_i) + E(s_{iw}) + E(\min.(s_{kw}, Q_w(Y) + WL_{iyw})) \qquad (7.8)$$

In Case 3 (Fig. 7.11(c)), PR_{imw} = P(T_i rejected by T_m at node w)

$= P(AB < AD < AC) = P(MR_m + s_{mw} > a_m) \; P(MR_m + s_{mw} > MR_i) \cdot$

$$P(MR_m + s_{mw} > t_{iw}) \; P(Q_w(Y) + WL_{iyw} > MR_m + t_{mw}) \qquad (7.9)$$

and W_{imw} = E(delay for T_i due to rejection by T_m at node w)

$$= E(s_{wi}) + E(MR_i) + E(r_{iw}) + E(\min.(MR_m + s_{mw}, Q_w(Y) + WL_{iyw})) \qquad (7.10)$$

We can now calculate the probability of restart for each transaction class. Consider Class 1 transactions T_1:

$$P_1 = P(\ T_1 \text{ is restarted }\ )$$

$$= 1 - P(\ T_1 \text{ not restarted }\ )$$

$$= 1 - \prod_{i \neq 1} \prod_{\alpha} P(\ T_1 \text{ not restarted by } T_i \text{ at node } \alpha\ )$$

since (1) the probability a transaction is restarted at the same node by different transactions are independent, and (2) the probability a transaction is restarted at different nodes by the same transaction (or by different transactions) are independent.

Fig. 7.2 shows at which nodes the transaction classes conflict. For example, T_1 and T_2 conflict at node 3. Equ.(7.7) gives

$$PR_{123} = P(s_{23} > a_2)\ P(s_{23} > MR_1)\ P(s_{23} > r_{13})\ P(Q_3(Z) + WL_{1Z3} > s_{23})$$

$$= \frac{\lambda_2}{\lambda_2 + \mu_{23}} \cdot \frac{1.957}{1.957 + \mu_{23}} \cdot \frac{\nu_{13}}{\nu_{13} + \mu_{23}} \cdot \frac{\mu_{23}}{\mu_{23} + 2.513} = .00042$$

where $\mu_{ij} = 1/E(s_{ij})$, $\nu_{ij} = 1/E(r_{ij})$, and it is assumed that MR_1 and $Q_3(Z) + WL_{1Z3}$ are exponentially distributed. Equ. (7.8) gives

$$W_{123} = E(s_{31}) + MR_1 + E(s_{13}) + E(s_{23}) = .609 \text{ sec.}$$

Similarly, we can find the probability a Class i transaction will be rejected by a Class j transaction at node k (i,j,k = 1,2,3,4,5) and the expected delay due to the rejection.

$$P_i = P(\ T_i \text{ is restarted }\ )$$

$$= 1 - \prod_{\substack{j \neq i \\ k=1,..,5}} P(\ T_i \text{ not restarted by } T_j \text{ at node } k\ )$$

$$W_i = E(\text{ delay for } T_j \text{ due to rejection }) = \sum_{\substack{j \neq i \\ k=1,..,5}} PR_{ijk} \cdot W_{ijk}$$

The following values of P_i and W_i were calculated:

$P_1 = .0483, \quad W_1 = .0102$ sec.

$P_2 = .0540, \quad W_2 = .0092$ sec.

$P_3 = .1325, \quad W_3 = .0421$ sec.

$P_4 = .0081, \quad W_4 = .0307$ sec.

$P_5 = .2171, \quad W_5 = .0607$ sec.

The average response time of the 5 Classes of transactions can now be calculated. For Class i transactions, average response time under Distributed Locking with Prioritized Transactions for Deadlock Prevention,

$RDLPT_i = MR_i + MW_i +$ delay due to rejection

$= MR_i + MW_i + P_i W_i$ (see Fig. 7.10)

Hence, $RDLPT_1 = .918$ sec.

$RDLPT_2 = .171$ sec.

$RDLPT_3 = .429$ sec.

$RDLPT_4 = .530$ sec.

$RDLPT_5 = .321$ sec.

7.4 SDD-1

In this example, we shall calculate the response time for the five classes of transactions under the SDD-1 Concurrency Control Algorithm. We are using the same notations as in section 7.1 and making the same assumpitons.

The volume of messages generated by SDD-1 is similar to that described in section 7.2. It is therefore assumed that the average delays on the communication channels are the same (as shown in Fig. 7.8).

Let $\bar{s}_{ij}$, $\bar{r}_{ij}$ denote respectively the average delay of a short and a long message between nodes i and j. Since transmission delays are assumed to be exponentially distributed, the parameters of the exponential distribution corresponding to short and long messages, are given by $u_{ij} = 1/\bar{s}_{ij}$ and $v_{ij} = 1/\bar{r}_{ij}$.

Let us first construct the conflict graph for our five classes of transactions The conflict graph (See Fig. 7.12) consists of nodes representing the readsets and writesets of the transaction classes, The links on the graph indicate potential conflict between the transactions. Therefore, two nodes are connected if at least one of them is a writeset and they have at least one file in common.

In SDD-1 [BSR80], the conflict graph is analyzed during database design and synchronization protocols are devised to maintain serializability. It is found that three protocols P1, P2, and P3 are necessary. A fourth protocol P4, is sometimes invoked to improve on the efficiency of the other three protocols. The SDD-1 protocol selection rules (Fig.7.13) state which prototcols should be invoked by which transactions.

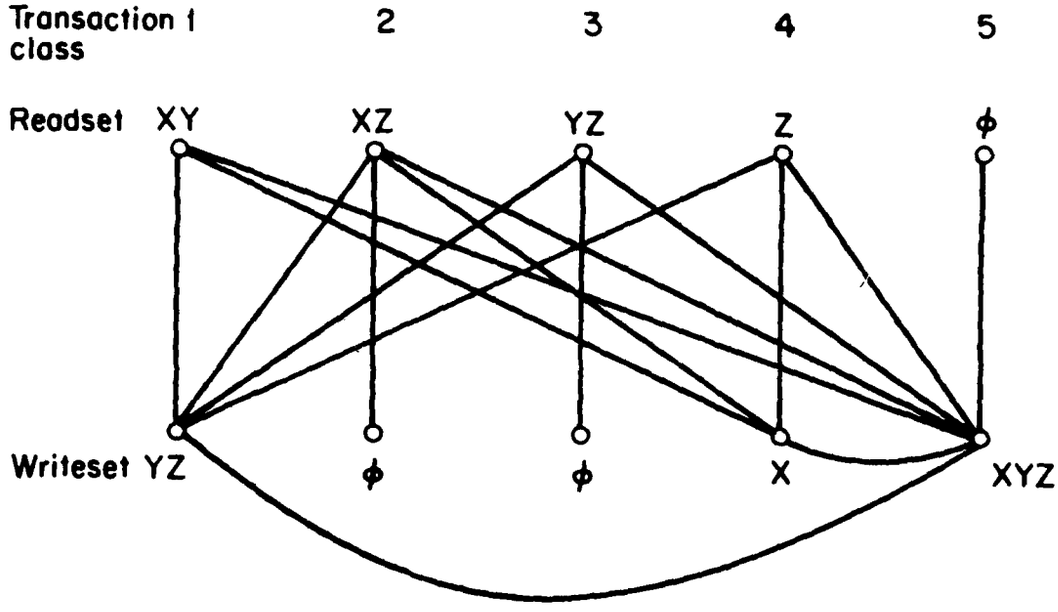
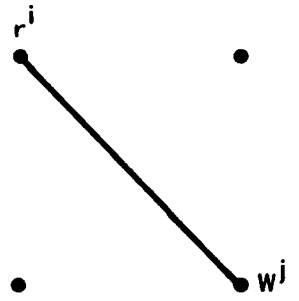
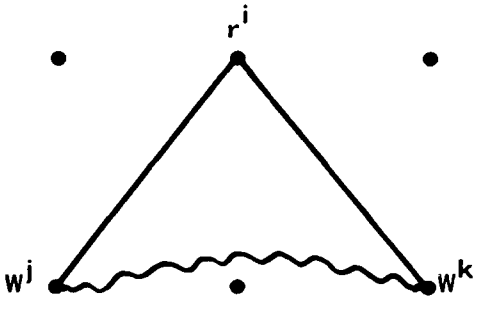


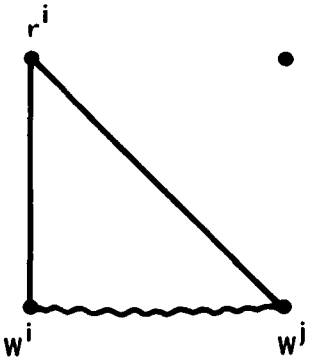
Figure 7.12 Conflict Graph for Transaction Classes in SDD-1 Example



(a) Transactions in class i must obey P1 with respect to transactions in class j



(b) Transactions in class i must obey P2 with respect to transactions in classes j and k



(c) Transactions in class i must obey P3 with respect to transactions in class j

Figure 7.13 SDD-1 Protocol Selection Rules (Adapted from [BSR80])

SDD-1 Protocol Selection Rules (adapted from [BSR80])

(1) For all classes i and j such that (r^i, w^j) is in the conflict graph, transactions in i must obey protocol P1 with respect to transactions in j(see Fig.7.13(a).

(2) For each cycle in the conflict graph the following hold:

(a) for all distinct classes i, j, k, if edges (r^i, w^j) and (r^i, w^k) lie on the cycle, then transactions in i must obey P2 with respect to transactions in j and k (see Fig.7.13(b)) and

(b) for all distinct classes i and j such that (r^i, w^i) and (r^i, w^j) lie on the cycle, then transactions in i must obey P3 with respect to transactions in j (see Fig.7.13(c)).

Briefly, these protocols serve the following purposes:

P1 Prevents read messages from one transaction that conflict with write messages from another transaction from being processed in different relative orders at different DMs.

P2 Prevents a read message from seeing write messages from two other transactions in reverse timestamp order.

P3 Prevents two transactions that read each other's output from both reading before either writes, i.e., prevents a classical race condition.

According to the SDD-1 protocol selection rules, and the conflict graph (Fig.7.12), it is necessary that:

Class 1 transaction runs P3 against Class 5 transactions

Class 2 transaction runs P2 against Classes 1 and 5

Class 3 transaction runs P2 against Classes 1 and 5

Class 4 transaction runs P2 against Classes 1 and 5

Class 5 transaction runs P3 against Class 5

The last two requirements are equivalent to:

Class 4 runs P2 against Class 1, and Class 4 runs P3 against class 5.

An inspection of Fig. 7.2 shows at which nodes these protocols are to be executed for each transaction.

Consider a Class 1 transaction T_1. It is running P3 against Class 5 transactions T_5. Thus when T_1 tries to read file Y at node 4, where T_5 is writing file Y, the protocol is invoked. The timestamp of T_1 must be smaller than that of file Y, in order for T_1 not to be rejected.

Let P_{ij}^k = P(T_i rejected by T_j at node k),

W_{ij}^k = time T_i has to wait at node k until its read condition against T_j is satisfied

D_{ij}^k = delay of T_i due to read rejection by T_j at node k.

P_i = P(T_i is rejected)

W_i = delay of T_i corresponding to query processing

D_i = delay of T_i due to read rejection

Therefore, $P_1 = P_{15}^4$ since T_1 only needs run the synchronization protocol against T_5 at node 4. Equ.(6.12) gives $P_1 = P_{15}^4 = P(s_{14} > a_5 + r_{54})$

$= \frac{\lambda_5}{\lambda_5 + u_{14}} \frac{v_{54}}{v_{54} + u_{14}} = .00325$, where a_5 = interarrival time of Class 5 transactions at node 5. (Recall that we assume read messages are short and write messages are long). Each rejection incurs additional delay = $D_1 = D_{15}^4$ round-trip delay from node 1 to node 4 = $s_{14} + s_{41}$ = .105sec. If not rejected, T_1 must wait at node 4 until its read condition is satisfied. Equ.(6.14) gives this expected wait as $E(W_{15}^4) = \frac{1}{v_{54}} + \frac{u_{14}}{u_{14}+\lambda_5} \cdot \frac{1}{\lambda_5}$ = 5.05 sec. This

wait can be reduced if node 5 send periodic nullwrites.

Suppose node 5 send nullwrites whenever the time since the last write message is greater than 1 second, then Equ.(6.28) and (6.29) give $P_1 = P_{15}^4$ = .0334 and $E(W_{15}^4)$ = .3974 sec.

Let $RSDD_i$ be the response time of transaction under SDD1, then

$RSDD_1$ = E(delay due to read rejection) + query processing delay + write delay

$= (\bar{s}_{14} + \bar{s}_{41})/(1 - P_1) + \bar{s}_{14} + E(W_1) + \bar{r}_{41} + \max.(\bar{r}_{12} + \bar{r}_{24} + \bar{s}_{43} + \bar{s}_{31}, + \bar{s}_{53} + \bar{s}_{31}) = .724$ sec.

where $1/(1 - P_1)$ is the expected number of rejections.

We next consider Class 2 transactions T_2. T_2 runs P2 against Classes 1 and 5. An inspection of Fig. 7.2 shows that T_2 runs P2 against T_1 at node 3, P2 against T_5 at node 2 and P2 against T_5 at node 3. Using Equ. (6.28) and (6.29), we find P_{21}^3 = .03128 and $E(W_{21}^3)$ = .3317 sec.

Similarly, P_{25}^3 = .02099

$E(W_{25}^3)$ = .361 sec,

$P_{25}^2 = 0$

$E(W_{25}^2)$ = .395 sec.

Therefore, from Equ.(III.1), P_2 = 1 - P(T_2 not rejected)

$= 1 - (1-P_{21}^3)(1 - P_{25}^3)(1 - P_{25}^2)$

$= .052$

D_2 = E(delay due to rejection)

$$= \frac{P_{21}^3(\bar{s}_{21}+\bar{s}_{12}) + P_{25}^3(\bar{s}_{23}+\bar{s}_{32})}{P_{21}^3 + P_{25}^3}$$

= .041 sec.

If T_2 is not rejected, then the delay corresponding to query processing, W_2 is given by Equ.(III.2) as follows:

$$W_2 = \max. \left[s_{23} + \max(W_{21}^3, W_{25}^3) + r_{32},\ s_{22} + W_{25}^2 + r_{22} \right]$$

$$\text{Therefore } E(W_2) \simeq \max. \left[\bar{s}_{23} + \max(E(W_{21}^3), E(W_{25}^3)) + \bar{r}_{32},\ \bar{s}_{22} + E(W_{25}^2) + \bar{r}_{22} \right]$$

$$= .447 \text{ sec.}$$

$$\text{Hence, } RSDD_2 = D_2/(1 - P_2) + E(W_2) = .490 \text{ sec.}$$

Consider Class 3 transactions T_3. T_3 runs P2 against T_1 and T_5 at node 3 and P2 against T_1 and T_5 at node 4.

$$P_3 = P(T_3 \text{ rejected})$$

$$= 1 - (1 - P_{31}^3)(1 - P_{31}^4)(1 - P_{35}^3)(1 - P_{35}^4)$$

$$= 1 - (1 - 0)(1 - .0910)(1 - 0)(1 - .0636) = .149$$

$$D_3 = E(\text{delay due to rejection})$$

$$= \frac{P_{31}^4(\bar{s}_{34} + \bar{s}_{43}) + P_{35}^4(\bar{s}_{34} + \bar{s}_{43})}{P_{31}^4 + P_{35}^4} = \bar{s}_{34} + \bar{s}_{43} = .105 \text{ sec.}$$

If T_3 is not rejected, then the delay corresponding to query processing W_3 is given by

$$W_3 = \max. \left[s_{33} + \max(W_{31}^3, W_{35}^3) + r_{33},\ s_{34} + \max(W_{31}^4, W_{35}^4) + r_{43} \right]$$

$$E(W_3) \simeq \max \left[\bar{s}_{33} + \max(E(W_{31}^3),\ E(W_{35}^3) + \bar{r}_{33},\ \bar{s}_{34} + \max(E(W_{31}^4), E(W_{35}^4)) + \bar{r}_{43} \right]$$

$$= .499 \text{ sec.}$$

$$\text{Hence, } RSDD_3 = D_3/(1 - P_3) + E(W_3) = .622 \text{ sec.}$$

Consider Class 4 transactions T_4. T_4 runs P2 against T_1 at nodes 1 and 5, P3 against T_5 at node 5.

P_4 = P(T_4 rejected)

$= 1 - (1 - P_{41}^{1})(1 - P_{41}^{5})(1 - P_{45}^{5})$

$= 1 - (1 - .1131)(1 - .0267)(1 - .0804) = .206$

D_4 = E(delay due to rejection)

$$= \frac{P_{41}^{1}(\bar{s}_{41}+\bar{s}_{14}) + P_{41}^{5}(\bar{s}_{45}+\bar{s}_{54}) + P_{45}^{5}(\bar{s}_{45} + \bar{s}_{54})}{P_{41}^{1} + P_{41}^{5} + P_{45}^{5}}$$

$= .116$ sec.

If T_4 is not rejected, then the expected delay corresponding to query processing

$E(W_4) = \max. \{s_{41} + E(W_{41}^{1}) + r_{14}, s_{45} + \max(E(W_{41}^{5}), E(W_{45}^{5})) + r_{54}\} = .542$ sec.

Hence, $RSDD_4 = D_4/(1 - P_4) + E(W_4) + \bar{r}_{43} + \bar{r}_{31} + \bar{r}_{12} + \bar{s}_{24} = .826$ sec.

Class 5 transactions do not have to observe any of the protocols, therefore

$RSDD_5$ = write delay

$= \max.(r_{53}+s_{35}, r_{53}+r_{31}+s_{13}+s_{35}, r_{52}+s_{21}+s_{13}+s_{35}, r_{54}+s_{45})$

$= .143$ sec.

7.5 Discussion of Numerical Examples

The results for the four examples described in sections 7.1 - 7.4 are summarized as follows:

Concurrency Control Algorithm \ Response times of transaction class	1	2	3	4	5	All Classes*
1. Centralized Two-phase Locking	.507	.220	.256	.775	.487	.441
2. Distributed Locking Ordered Queues for Deadlock Prevention	4.02	.509	.692	3.28	4.41	2.874
3. Distributed Locking Prioritized Transactions for Deadlock Prevention	.918	.171	.429	.530	.321	.522
4. SDD-1	.724	.490	.622	.826	.143	.543

Although we have used an arbitrary example to compare the different algorithms and any conclusions drawn based on these results may not apply in general, it does seem obvious that Algorithm 2 gives the worst response times. This is mainly because of the requirement that files have to be locked in a specific order. This requirement does not allow much concurrency.

The numerical results do not let us distinguish the performance of Algorithm 1, 3 and 4. Which algorithm is better depends on the network topology and such database parameters as arrival rates of transactions, size

---

*The response time for all classes is a weighted average (by transaction class arrival rate) of the response time for each individual class.

of writesets, readsets, etc. For example, if the transaction arrival rates increase, Algorithm 1 will give longer response times since the central node becomes a bottleneck. This does not happen in this example.

In general, we believe that the more concurrency an algorithm allows, the smaller its average response time. Thus, we would expect SDD-1 and Distributed Locking to give better response times than Centralized Locking.

CHAPTER 8

CONCLUSIONS

8.1 Conclusions

In this thesis, we have developed a performance model of a distributed database system, which can be used as a tool to compare the performance of different concurrency control algorithms.

We started by developing a network of queues model of the communication subnetwork. We have originally attempted to employ Jackson's Model but have concluded that Jackson's Model is inadequate for our purposes. The Independent Queues Model that we employed in this thesis makes somewhat stronger assumptions than Jackson's Model, but has more flexibility and approximates better a real communication subnetwork. Modelling the communication subnetwork accurately is important because one of the major costs of operating a DDB is the communication delay.

We found that in a general DDB, concurrency control algorithms could not be modelled accurately without taking into consideration the particular query processing strategy employed. Previous authors have gotten around the problem by assuming a fully redundant database. We found this assumption unacceptable and therefore attempted to develop a new query processing strategy that is easy to analyze. Our efforts resulted in the MST and the MDT Algorithms which are not only easy to analyze but also easy to implement.

Having modelled the conflicts among different transactions in the DDB for the resources of the communication subnetwork by the Independent Queues Model, we then developed conflict models to analyze the conflict among transactions for the resources of the database management system. A different conflict model must be developed for each concurrency control

algorithm. Fortunately, although the literature is full of concurrency control methods, most are variations fo two major approaches, namely two-phase locking and timestamp ordering. Four different conflict models were developed: Centralized Two-Phase Locking with Deadlock Detection, Distributed Two-Phase Locking with Ordered Queues for Deadlock Prevention, Distributed Two-Phase Locking with Prioritized Transactions for Deadlock Prevention, and SDD-1.

Four numerical examples using a common communication subnetwork were used to demonstrate how our performance model could be used to analyze these four concurrency control algorithms.

One would hope that at the end of a study such as this, one can draw some conclusions as to which concurrency control algorithm is the best. Unfortunately, the most general conclusion we can draw is that which algorithm is better depends very much on the particular communication subnetwork and the DDB system.

8.2 Further Research

In this thesis we have touched upon many different aspects of a DDB Management system. Due to time constraints, we have not been able to study all the different problems in as much depth as we would like to. There are a number of open problems, listed below are some suggestions for further research:

(1) Our study realizes that communication links and computers are not perfectly reliable and incorporates some of the features database systems used to guard against such failures, such as two-phase commit. However, we did not analyze the impact of such failures

in terms of extra delay.

(2) Our conflict models assume exponential end-to-end transmission delays. New conflict models using other distribution of end-to-end delays can be developed in a similar fashion.

(3) We believe that the MST and MDT Algorithms for query processing are easy to analyze and to implement. However, they suffer from very strict assumptions. Maybe the two algorithms can be extended by relaxing some of these assumptions.

(4) The MST and MDT Algorithms, because of their unrealistic assumptions, are actually heuristics, as is Wong's Algorithm [WONG77]. It should be interesting to compare them.

(5) One important part of model development that we have not studied in this thesis is that of model validation. Unfortunately, since there are no operating commerical systems, the only way to validate our performance model is by simulation, which is expensive.

Appendix I Finding the pdf of x, given x < y where x, y are exponential random variables.

Let $f_x(x_o) = \lambda_1 e^{-\lambda_1 x_o}$ $\quad x_o \geq 0$

$f_y(y_o) = \lambda_2 e^{-\lambda_2 y_o}$ $\quad y \geq 0$

Event A : $x < y$

$$f_{x,y|A}(x_o,y_o|A) = \begin{cases} \lambda_1\lambda_2 e^{-\lambda_1 x_o} e^{-\lambda_2 y_o} / P(A) & \text{if } (x_o,y_o) \text{ in A} \\ 0 & \text{if } (x_o,y_o) \text{ not in A} \end{cases}$$

Therefore, $f_{x|A}(x_o|A) = \int_{x_o}^{\infty} f_{x,y|A}(x_o,y_o|A)\, dy_o$

$$= \int_{x_o}^{\infty} \frac{\lambda_1\lambda_2 e^{-\lambda_1 x_o} e^{-\lambda_2 y_o}}{\lambda_1/(\lambda_1 + \lambda_2)}\, dy_o$$

$$= (\lambda_1 + \lambda_2)\, e^{-(\lambda_1 + \lambda_2)\, x_o} \qquad x_o \geq 0$$

which is the same as the pdf of min.(x,y).

Appendix II Finding the maximum of queueing times at several M/M/1 queues

Let w_i be the waiting time at queue i with arrival rates λ_i, service rate μ_i, utilization $\rho_i = \lambda_i / \mu_i$, and $x = \max_i w_i$, then

$$F_x(x_o) = P(x \le x_o) = \prod_{i=1}^{N} P(w_i \le x_o)$$

$$= \prod_i \left(1 - \rho_i e^{-\mu_i(1-\rho_i)x_o} \right)$$

$$f_x(x_o) = \frac{d}{dx_o} F_x(x_o) = \sum_i \rho_i \mu_i (1 - \rho_i) e^{-\mu_i(1-\rho_i)x_o} \prod_{j \neq i} \left[1 - \rho_j e^{-\mu_j(1-\rho_j)x_o} \right]$$

$$E(x) = \int_0^\infty (1 - F_x(x_o))\, dx_o$$

For the special case that $x_2 = \max(w_1, w_2)$, we have

$$E(x_2) = \int_0^\infty \left[1 - (1 - \rho_1 e^{-\mu_1(1-\rho_1)x_o})(1 - \rho_2 e^{-\mu_2(1-\rho_2)x_o}) \right] dx_o$$

$$= \int_0^\infty \left[\rho_2 e^{-\mu_2(1-\rho_2)x_o} + \rho_1 e^{-\mu_1(1-\rho_1)x_o} - \rho_1\rho_2 e^{-(\mu_2(1-\rho_2)+\mu_1(1-\rho_1))x_o} \right] dx_o$$

$$= \frac{\rho_2}{\mu_2(1-\rho_2)} + \frac{\rho_1}{\mu_1(1-\rho_1)} - \frac{\rho_1\rho_2}{\mu_2(1-\rho_2) + \mu_1(1-\rho_1)}$$

In general, if $x_k = \max(w_1, w_2, \ldots, w_k)$, we see that

$$E(x_k) = \sum_{i=1}^{k} \frac{\rho_i}{\mu_i(1-\rho_i)} - \sum_{i=1}^{k} \sum_{j \neq i} \frac{\rho_i\rho_j}{\sum_{\ell=i,j} \mu_\ell(1-\rho_\ell)}$$

$$\sum_{i=1}^{k} \sum_{j \neq i} \sum_{\ell \neq j} \frac{\rho_i\rho_j\rho_\ell}{\sum_{m=i,j,\ell} \mu_m(1-\rho_m)} - \cdots \pm \frac{\rho_1\rho_2\cdots\rho_k}{\sum_{i=1}^{k} \mu_i(1-\rho_i)}$$

Appendix III SDD1: General Case

In this Appendix, we derive the probability of read rejection and expected wait until read condition is satisfied when there are more than two conflicting transaction classes.

Let T_i, T_j, ... T_ℓ denote transactions,

I, J, ... L denote nodes,

N(i), N(j) ...N(ℓ) denote the originating node of transactions i, j, ...ℓ .

a_i' = interarrival time of T_i at N(i)

t_{IJ} = transmission delay between nodes I and J

P_{ij}^L = P(T_i rejected by T_j at node L)

W_{ij}^L = (period of time T_i has to wait at node L for its read condition to be satisfied when it is running protocol P3 against T_j T_i is not read rejected)

Suppose T_i is running protocol P3 against T_j, $j \in G(J)$ at node J, T_k, $k \in G(K)$ at node K, ...T_m, $m \in G(M)$ at node M. G(J) denotes the set of transactions that T_i conflicts with at node J.

$$P(T_i \text{ is rejected}) = 1 - P(T_i \text{ not rejected})$$

$$= 1 - \prod_{\alpha=I,J,\ldots L} P(T_i \text{ not rejected at node } \alpha)$$

$$= 1 - \prod_{\alpha=I,J,\ldots L} \prod_{\beta \in G(\alpha)} (1 - P_{i\beta}^{\alpha}) \qquad \text{(III.1)}$$

(Query Processing Delay | T_i not rejected)

= (Wait for all read conditions to be satisfies | T_i not rej.)

$$= \max_{\alpha=I,J,\ldots L} \left(t_{N(i)\alpha} + \max_{\beta \in G(\alpha)} (W_{i\beta}^{\alpha}) + t_{\alpha N(i)} \right) \qquad \text{(III.2)}$$

where P_{ij}^L and $E(W_{ij}^L)$ are given by Equ.(6.28) and (6.29) respectively.

Let $A = 1 - e^{-\lambda_j S}(\lambda_j + 1)$,

$$B = \frac{\lambda_j}{A} \frac{\mu_{iL} - e^{-\lambda_j S}(\mu_{iL} + \lambda_j - \lambda_j e^{-\mu_{iL} S})}{\mu_{iL}(\mu_{iL} + \lambda_j)}$$

then $P_{ij}^{L} = P(t_{iL} > a_j' + t_{jL})$

$$= \mu_{jL} B / (\mu_{iL} + \mu_{jL})$$

and $E(W_{ij}^{L}) = 1/\mu_{jL} + (1/\lambda_j - \lambda_j S^2 e^{-\lambda_j S} / 2A)(1 - B)$

Appendix IV Finding the expected value of the maximum of several exponentials.

Let y_i's be N exponentials with means $1/\lambda_i$. We first find the pdf of $x_k = \max_{i=1,k} (y_i)$

$$F_{x_k}(x) = P(x_k \leq x) = \prod_{i=1}^{k} P(y_i \leq x) = \prod_{i=1}^{k} (1 - e^{-\lambda_i x})$$

$$f_{x_k}(x) = \frac{d}{dx} F_{x_k}(x) = \sum_{i=1}^{k} \lambda_i e^{-\lambda_i x} \prod_{j \neq i} (1 - e^{-\lambda_j x})$$

Hence, $E(x_k) = \int_0^\infty (1 - F_{x_k}(x))dx = \int_0^\infty \left[1 - \prod_{i=1}^{k} (1 - e^{-\lambda_i x})\right] dx$

$$= \sum_{i=1}^{k} \frac{1}{\lambda_i} - \sum_{i=1}^{k} \sum_{j \neq i} \frac{1}{\lambda_i + \lambda_j} + \sum_{i=1}^{k} \sum_{j \neq i} \sum_{\ell \neq j} \frac{1}{\lambda_i + \lambda_j + \lambda_\ell} - \cdots\cdots$$

$$(\pm) \frac{1}{\sum_{i=1}^{k} \lambda_i}$$

This is similar to the derivation in Appendix II and will not be repeated here.

REFERENCES

[AD76] Alsberg, P.A., and Day, J.D., "A Principle for Resilient Sharing of Distributed Resources", Proc. 2nd Int. Conference on Software Engineering, October 1976.

[BCMP75] Baskett, F., Chandy, K.M., Muntz, R.R., and Palacios, F.G., "Open, Closed and Mixed Networks of Queues with Different Classes of Customers," J. Assoc. Comput. Mach., Vol. 22 (1975). pp. 248-260

[BG80] Bernstein, P.A., Goodman, N., "Fundamental Algorithms for Concurrency Control in Distributed Database Systems," Computer Corporation of America, February 15, 1980.

[BS80] Bernstein, P.A., and Shipman, D., "The Correctness of Concurrency Mechanisms in a System for Distributed Databases (SDD-1)," ACM Trans. on Database Systems, Vol. 5, No. 1, March 1980.

[BSR80] Bernstein, P.A., Shipman, D., and Rothnie, J., "Concurrency Control in a System for Distributed Databases (SDD-1)," ACM Trans. on Database Systems, Vol. 5, No. 1, March 1980.

[BSW79] Bernstein, P.A., Shipman D., and Wong, W.S., "Formal Aspects of Serializability in Database Concurrency Control," IEEE Trans. on Software Engineering, Vol. SE-5, No. 3, May 1979.

[BURK56] Burke, P.J., "The Output of a Queueing System," Operations Research, Vol. 4, 1956, pp. 699-704.

[CALO80] Calo, S.B., " Message Delays in Repeated-Service Tandem Connections," IBM Research Report RC8175, Thomas J. Watson Research Center, March 25, 1980.

[CHIU79] Chiu, W.D.M., "Optimal Query Interpretation for Distributed Databases," Ph.D. Dissertation, Division of Applied Sciences, Harvard University, December 1979.

[CHRI75] Christofides, Nicos, Graph Theory: An Algorithmic Approach, Academic Press, New York, 1975.

[CODD70] Codd, E.F., "A relational model of data for large shared data banks," Comm. ACM, Vol. 13, June 1970, pp.377-387.

[COX55] Cox, D.R., "A Use of Complex Probabilities in the Theory of Stochastic Processes," Proc. Cambridge Philosophical Society, 51, 1955, pp. 313-319.

[DATE77] Date, C., An Introduction to Database Systems, 2nd Ed., Addison-Wesley, 1977.

[Drak67] Drake, A.W., Fundamentals of Applied Probability Theory, McGraw-Hill, 1967.

[EGLT76] Eswaran, K.P., Gray, J.N., Lorie, R.A. and Traiger, I.L.," The Notions of Consistency and Predicate Locks in a Database System," Comm. ACM, Vol. 19, No. 11, November 1976.

[ELLI77] Ellis, C.A., " A Robust Algorithm for Updating Duplicate Databases," Proc. 2nd Berkeley Workshop on Distributed Databases and Computer Networks, May 1977.

[GALL77] Gallager, R.G.," A Minimum Delay Routing Algorithm Using Distributed Computation," IEEE Trans. on Comm., Vol. COM-25, No. 1, January 1977, pp. 73-85.

[GARC79] Garcia-Molina, H.," Performance of Update Algorithms for Replicated Data in a Distributed Database," Ph.D. Dissertation, Computer Science Department, Stanford University, June 1979.

[GLPT75] Gray, J.N., Lorie, R.A., Putzolu, G.R. and Traiger, I.L.," Granularity of Locks and Degrees of Consistency in a Shared Data Base," IBM Res. Rep. RJ1654, September 1975.

[HAKI71] Hakimi, S.L.," Steiner's Problem in Graphs and Its Implications," Networks, 1, 1971, pp. 113-133.

[HY79] Hevner, A.R. and Yao, S.B.," Query Processing in Distributed Databases," IEEE Trans. on Software Eng., Vol. SE-5, No. 3, May 1979.

[JACK57] Jackson, J.R.," Networks of Waiting Lines," Operations Research, Vol. 5, 1957, pp. 518-521.

[KC74] King,P.F. and Collmeyer, A.J.," Database Sharing --- An Efficient Mechanism for Supporting Concurrent Processes," Proc. 1974 NCC, AFIPS Press, Montvale, New Jersey, 1974.

[KLEI64] Kleinrock, L., Communication Nets: Stochastic Message Flow and Delay, McGraw-Hill, New York, 1964.

[KLEI75] _____, Queueing Systems, Volume 1, John Wiley & Sons, 1975.

[KLEI76] _____, Queueing Systems, Volume 2, John Wiley & Sons, 1976.

[KP79] Kung, H.T. and Papadimitriou, C.H.," An Optimality Theory of Concurrency Control for Databases," Proc. ACM SIGMOD Conf., Boston, Mass., May 1979, pp. 116-126.

[LITT61] Little, J.D.C., " A Proof of the Queueing Formula L = λW," Operations Research, Vol. 9, 1961, pp. 383-387.

[MOLI27] Molina, E.C.," Application of the Theory of Probability to Telephone Trunking Problems," Bell Systems Tech. J., 6, 1927, pp. 461-494.

[PAPA79] Papadimitriou, C.H.," Serializability of Concurrent Updates," Journal of the ACM, Vol. 26, No. 4, October 1979, pp. 613-653.

[RG77] Rothnie, J.B. and Goodman, N.," A Survey of Research and Development in Distributed Database Management," Proc. 3rd Intl. Conf. on Very Large Databases, IEEE, 1977, pp. 48-62.

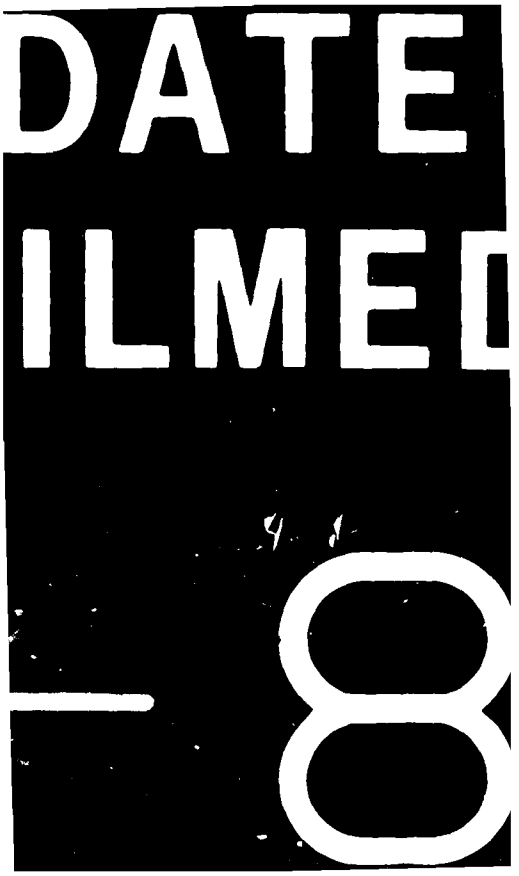
[ROTH80] Rothnie, J.B., Bernstein, P.A., Fox, S.A., Goodman, N., Hammer, M.M., Landers, T.A., Reeve, C.L., Shipman, D.W. and Wong, E., " Introduction to a System for Distributed Databases," ACM Trans. on Database Systems, Vol. 5, No. 1, March 1980.

[RS77] Ries, D.R. and Stonebraker, M., " Effects of Locking Granularity in a Database Management System," ACM Trans. on Database Systems, Vol. 2, No. 3, September 1977, pp. 233-246.

[RSL78] Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M.," System Level Concurrency Control for Distributed Database Systems," ACM Trans. on Database Systems, Vol. 3, No. 2, June 1978, pp. 178-198.

[THOM79] Thomas, R.H.," A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," ACM Trans. on Database Systems, Vol. 4, No. 2, June 1979, pp. 180-209.

[WONG77] Wong, E.," Retrieving Dispersed Data from SDD-1: A System for Distributed Databases," Rep. CCA-77-03, Computer Corp. of America, March 15, 1977.


DATE
ILME
8